Criojo: A Pivot Language for Service-Oriented Computing - The Introspective Chemical Abstract Machine

Hervé Grall, Mayleen Lacouture

▶ **To cite this version:**

Hervé Grall, Mayleen Lacouture. Criojo: A Pivot Language for Service-Oriented Computing - The Introspective Chemical Abstract Machine. 2012. hal-00676083v2

HAL Id: hal-00676083
https://hal.science/hal-00676083v2

Preprint submitted on 7 Mar 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Criojo: A Pivot Language for Service-Oriented Computing

The Introspective Chemical Abstract Machine

Hervé Grall and Mayleen Lacouture

ASCOLA Research Team (Mines de Nantes–INRIA, LINA), France

Table of Contents

Abstract. Interoperability remains a significant challenge in service-oriented computing. After proposing a pivot architecture to solve three interoperability problems, namely adaptation, integration and coordination problems between clients and servers, we explore the theoretical foundations for this architecture. A pivot architecture requires a universal language for orchestrating services and a universal language for interfacing resources. Since there is no evidence today that Web Services technologies can provide this basis, we propose a new language called Criojo and essentially show that it can be considered as a pivot language. We formalize the language Criojo and its operational semantics, by resorting to a chemical abstract machine, and give an account of formal translations into Criojo: in a distributed context, we deal with idiomatic languages for four major programming paradigms: imperative programming, logic programming, functional programming and concurrent programming.

1 Introduction

Assume you want to automatize the management of your photos, by using Web photos management systems, like Picasa and Flickr. You may quickly face interoperability problems, namely *adaptation*, *integration* and *coordination* problems. Indeed, Picasa and Flickr are based over distinct interfaces, not only from a functional point of view, with distinct resource models, differently organizing photos, but also from a communicational one, since Flickr provides both Restful and WS* services, the mainstream technologies for Web services, while Picasa only provides Restful services. Therefore, an *adaptation* is needed when a client application that orchestrates Picasa services must evolve to orchestrate Flickr services, or conversely; or even when it must evolve from a Restful interface to a WS* interface, in the case of Flickr. An *integration* is needed when the client application must orchestrate both Picasa and Flickr services. A *coordination* is needed when two scripts, possibly written in distinct languages, must cooperate to orchestrate services provided by one system.

The Flickr vs Picasa scenario is summarized in Fig. 1. In the center, the Restful and WS* interfaces, provided by Flickr and Picasa, manage the resources (photos) on the right side. On the left side, you have some of the possible orchestration languages used to communicate with the interfaces. At first glance is clear the combinatorial explosion for communications from multiple languages to multiple interfaces.

In this paper, after proposing a pivot architecture to solve these problems, we explore the theoretical foundations for this architecture. A pivot architecture essentially requires a pivot language, which is universal both for orchestrating services and for interfacing resources. Since there is no evidence today that WS* or Restful technologies can provide the basis for a pivot architecture, we propose a new language called Criojo as a pivot language. Our contributions are as follows.

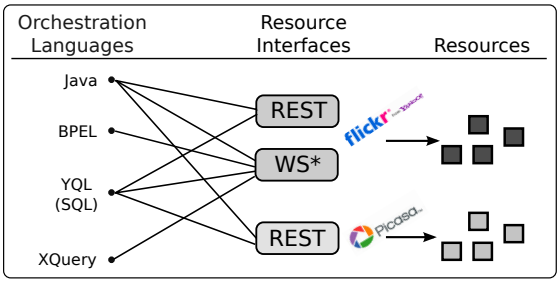

Fig. 1. Problem Communication from Multiple Languages to Multiple Interfaces

– We recall how a pivot architecture solves the problems of adaptation, integration and coordination. The solution requires three specific abilities for the associated pivot language:

- Defining wrappers for resources,
- Translating any orchestration language,
- Encoding the Adapter, Mediator and Facade patterns [15, pp. 139, 273, 185], used to adapt, integrate and coordinate.

We deduce from these requirements a specification of the pivot language, in the context of service-oriented computing.

– We formalize the language Criojo and its operational semantics, by resorting to a distributed chemical abstract machine. A program in Criojo actually corresponds to (i) the description of the collaboration between agents and (ii) the syntactic description of the reduction rules specific to the chemical abstract machine associated to each agent. Agents float in a chemical soup, which represents the pervasive infrastructure of a network (like Internet) where agents produce and consume messages. An agent acts as a server interfacing resources or as an orchestrator. It can be a black box, implemented in impure Criojo, which means that another language is also used, or an autonomous process executing a program written in pure Criojo. Thus, Criojo exists in two distinct flavors, pure and impure. To fulfill the first requirement, we simply define a wrapper of a resource as a black box server written in impure Criojo: therefore the language Criojo can provide in its impure flavor a universal language for interfacing resources.

– We give an account of the formal translation of four idiomatic languages into pure Criojo while considering a distributed context:

- Imperative Programming: a variant of Dijkstra's language of guarded commands,
- Logic Programming: Datalog with negation,
- Functional Programming: λ-calculus with inductive types and fixed point operator (Gödel's System T),
- Concurrent Programming: π-calculus.

These translations aim at substantiating the argument that the language Criojo can be the target language for compiling from any orchestration language, assumed to be represented by the four idiomatic languages (second requirement)

and allows adapters, mediators and facades to be encoded (third requirement). Thus the language Criojo can provide in its pure form a universal language for orchestrating services. This ability results from a powerful extension of standard chemical abstract machines: a program in pure Criojo can introspect the local chemical solution. Chemical abstract machines become *introspective*.
From a theoretical perspective, the language Criojo can therefore be seen as an attempt to combine different computational paradigms, such as imperative, logic (for queries), functional and concurrent programming in a clean, uniform, and effective way. From a practical perspective, an alternative to our proposal of a new language could be chosen: just select an existing language enough expressive and experiment it as a pivot language. But in our opinion this approach would have two drawbacks. First, showing the practicability of the solution in concrete cases would probably require an excessive implementation effort. Second, the experimentation would not emphasize the concepts that are essential for designing a pivot language. Thus our proposal follows a more economical and more foundational approach.

2 Specification of the Pivot Language

As seen in the example of Picasa and Flickr, the absence of a unified model for service-oriented computing leads to interoperability issues, namely *adaptation*, *integration* and *coordination* problems. To solve these problems, we have proposed a pivot architecture [24], as shown in Fig. 2. On the left side of the diagram, scripts written in existing orchestration languages, like BPEL or Java, are compiled into the pivot language, here Criojo. On the right side of the diagram, wrappers implemented in impure Criojo allow the interaction of the compiled scripts with the Restful and WS* interfaces proposed by Picasa and Flickr. In the middle of the diagram, we use design patterns to solve the adaptation, integration and coordination problems. The adaptation problem is solved with the Adapter pattern: an adapter built between the client and the new service provider allows to switch from one service provider to another without modifying the client. The integration problem is solved with the Facade pattern: an intermediate component built between the client and the two service providers offers a common representation for the two resource models. Finally, the coordination problem is solved with the Mediator pattern: a mediator component allows the coordination of two or more scripts by combining their results.

However, the solution relies on three assumptions for the pivot language Criojo: (i) that any orchestration language can be compiled into Criojo, (ii) that Criojo can interact with different resource interfaces, and (iii) that the design patterns used to solve interoperability issues can be encoded in Criojo. We turn these assumptions into three requirements for the pivot language.

Universality for Compiling In order to compile scripts written in different orchestration languages into Criojo, we need a multi-paradigm language. Concretely, the pivot language must support compilation from imperative languages like Java, functional languages like XQuery, concurrent languages like

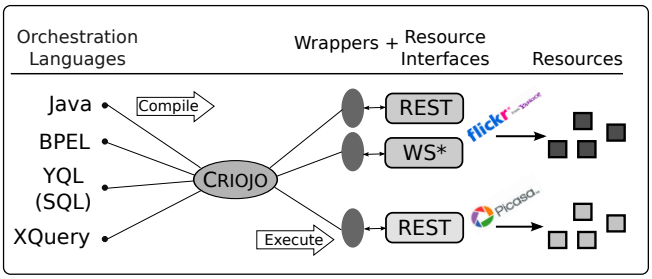

Fig. 2. The Pivot Architecture

BPEL, and logic languages like YQL or SQL, following an approximate classification since each language also presents features from other paradigms.

Universality for Interfacing Service interfaces differ not only from a functional point of view, but also from a communicational one. A universal language for representing resources is therefore required, as well as a middleware layer.

Expressivity The pivot language must be enough expressive to allow the different design patterns to be encoded. However, as we consider that this last requirement derives from the first one, we do not deal with it in the following. The interested reader can find examples with an earlier version of Criojo in a preceding article [24, Sect. 5].

Finally, we also add requirements specific to the context of service-oriented computing. As exemplified by mainstream technologies for Web services, service-oriented computing is an efficient solution to organise the exchange of messages in a network-based architecture, by making agents provide services. These agents are used not only to manipulate local resources, in response to requests, but also to remotely call other services, following a specific orchestration. Thus, each agent can play two roles, a *server*, as in the first case, and an *orchestrator*, which is also a client of other services, and often implements a business process, as in the second case. We have identified four requirements resulting from service orientation: message passing, channel mobility, scope management and Black Box principle.

Message Passing Traditionally, distributed systems are represented as sets of autonomous agents that execute concurrently and interact with each other. There are two classes of models [25]: message-passing models and the other ones, including models based on shared memory and on synchronous communication between sequential processes. Clearly, service-oriented computing, where agents communicate with each other by exchanging messages [22], requires to choose a message-passing model for the pivot language.

Channel Mobility Dynamic binding is necessary for service discovery and dynamic routing. Indeed, during an execution, the network topology often needs to evolve: an agent needs to discover another agent that it does not know initially. The π-calculus provides an elegant solution to the problem of dynamic binding, by allowing the mobility of communication channels: messages convey not only

values but also channels [32, p. 1]. Thus, we require channel mobility for the pivot language.

Scope Management The collaborations between agents are often organized around sessions. A session is identified by a token, with a particular scope: the token is generated when the session starts, and then shared between the agents participating to the session. Scope management is therefore required in the pivot language.

Black Box Principle To promote interoperability, service-oriented computing adheres to the Black Box principle: agents hide the implementation details of the services that they provide and use the services that they require only through their interfaces. Thus, the pivot language must allow an agent implementation to be abstracted away.

3 The Introspective Chemical Abstract Machine and its Language

In the section, we describe the syntax and the semantics of the pivot language Criojo. It can be described as the language associated to a chemical abstract machine dedicated to service-oriented computing. Finally, we define a notion of program equivalence, based on weak bisimulations.

Before the formalization, we start by a small example with a client and a server. The server provides a channel `ping` while the client provides a channel `pong` to get the response. The server also manages a local counter: when it receives a request over channel `ping`, it sends the current value of the counter to the client and increments the counter. The whole collaboration can be described as follows.

$$\texttt{Client}.(\texttt{Begin}) \parallel \texttt{Server}.(\texttt{Counter(0)})$$

The initial state of the client contains a unique internal message, `Begin`, whereas the initial state of the server contains a unique message `Counter(0)`, giving the initial value `0` of the counter. The behavior of the client is described by the following rules.

$$\texttt{Begin} \to \texttt{ping}(\texttt{pong}), \texttt{End} \qquad \texttt{pong}(n) \to \texttt{Print}(n)$$

It first sends the request to the server, then waits for the response, and finally prints the value received. The behavior of the server is described as follows.

$$\texttt{Counter}(n), \texttt{ping}(k) \to \texttt{Counter}(n+1), k(n)$$

The server indefinitely replies to requests by sending the value of the counter and incrementing its value. This simple example highlights some essential concepts: (i) a program in Criojo describes a collaboration and the behaviors of the agents involved in the collaboration.; (ii) there are two kinds of messages, internal ones like $\texttt{Print}(n)$ and external ones like $\texttt{pong}(n)$; (iii) channels are also values, like `pong`; (iv) a name server, like a counter, allows names to be locally managed.

3.1 Syntax and Semantics

The starting point for the formal definition of the pivot language Criojo is Berry and Boudol's chemical model [2], based on a chemical abstract machine (or cham). Thanks to the associativity and commutativity properties of multisets, their chemical model has been immediately acknowledged as an elegant formalism for concurrency, as shown for instance by Milner, who gave a new way of formulating the π-calculus [28]. The model is also well suited to distribution, and especially message-passing models. Indeed, chemical solutions can be organized in a hierarchy. Thus, any solution can contain cells, defined as membranes encapsulating sub-solutions, and which can be distributed. Cells evolve in a (truly) concurrent way: indeed chemical reactions, which are always local to a solution, can be performed in parallel, provided that they involve disjoint sets of molecules or cells. The communication between chemical solutions is also local: to migrate, a molecule need to move from a sub-solution to the airlock of the membrane, and then after reaction to the outer solution, or in the reverse direction.

After this general description, we now define the cham dedicated to service-oriented computing.

The molecules of the cham are messages, corresponding to requests and responses. A message is a value, defined as a term over some algebraic signature, and conveyed by a channel. To ensure channel mobility, channels belong to the signature as constants, and can therefore be considered as terms.

$$\text{Messages: } k(v) \qquad k \in \mathcal{K} \text{ (Set of Channels)}, v \in \mathcal{V} \text{ (Set of Values)}, \mathcal{K} \subseteq \mathcal{V}$$

In the following, the algebraic signature is left implicit. It is often tacitly assumed to contain tuple constructors, which are often omitted. For instance, we write $k(v_1, \ldots, v_n)$ instead of $k(c_n(v_1, \ldots, v_n))$, where c_n is the constructor of n-tuple. The cells of the cham are agents. Each agent a provides some channels, which together form the set $\mathcal{K}(a)$. It consumes incoming messages, produces outgoing messages and updates its state.

A program in Criojo first defines a collaboration between agents, as defined as follows.

Collaboration	$c ::= a.\,\sigma$	Agent a with State σ
	$\mid c \parallel c$	Collaborations in Parallel
Agent	$a \in \mathcal{A}$	Set of Agents
State	$\sigma \in \Sigma$	Set of States

We assume that the agent identifiers are pairwise distinct and that the sets $\mathcal{K}(a)$ of provided channels are pairwise disjoint. The collaboration is then deployed in the top-level chemical solution, called the web. Finally, the cham makes the web evolve, by performing chemical reactions. The web is defined as a multiset, as

follows.

Web	$\Omega ::= \langle\, \overrightarrow{\omega} \,\rangle$	Multiset of Web Entities ω
Entity	$\omega ::= c$	Collaboration to be Deployed
	$\mid a[\Lambda]$	Agent a with Local Solution Λ
	$\mid a.\,\alpha[\Lambda]$	Agent a with Local Solution Λ and Airlock α
	$\mid k(v)$	Message in Transit
Local Solution	$\Lambda ::= \langle\, \sigma \,\rangle \uplus \langle\, \overrightarrow{k(v)} \,\rangle$	State and Multiset of Messages
Airlock	$\alpha ::= \sigma \mid k(v)$	State or Message

It contains collaborations to be deployed, active agents, each one with a state, a local solution, defined as a multiset of messages, and possibly an airlock containing the initial state or a migrating message. The semantics of the cham is operational, defining a reduction relation. The generic reduction rules are described in Table 1. First, there are two kinds of inference rules, [CHEMICAL] and [MEMBRANE], which are laws, that is to say rules common to all chams: they allow reductions to locally occur in any solution. Second, there are specifics rules, for deployment and for communication, as well as the associated law [REACTION], allowing the specific rules (which are rule schemata) to be instantiated. The de-

[CHEMICAL]	$\dfrac{\Omega_1 \to \Omega_2}{\Omega_1 \uplus \Omega \to \Omega_2 \uplus \Omega}$	$\dfrac{a \vdash \Lambda_1 \to \Lambda_2}{a \vdash \Lambda_1 \uplus \Lambda \to \Lambda_2 \uplus \Lambda}\ (\Lambda = \langle\, \overrightarrow{k(v)} \,\rangle)$
[MEMBRANE]	$\dfrac{a \vdash \Lambda_1 \to \Lambda_2}{\langle\, a[\Lambda_1] \,\rangle \to \langle\, a[\Lambda_2] \,\rangle}$	$\dfrac{a \vdash \Lambda_1 \to \Lambda_2}{\langle\, a.\,\alpha[\Lambda_1] \,\rangle \to \langle\, a.\,\alpha[\Lambda_2] \,\rangle}$
[DEPLOYMENT]	$c_1 \parallel c_2 \to c_1, c_2$	$a.\,\sigma \to a.\,\sigma[\langle\,\rangle]$
[IN]	$k(v), a[\Lambda] \to a.\,k(v)[\Lambda] \quad \big(k \in \mathcal{K}(a)\big)$	
[IN-AIRLOCK]	$a.\,k(v)[\Lambda] \to a[\langle\, k(v) \,\rangle \uplus \Lambda] \quad \big(k \in \mathcal{K}(a)\big)$	
[OUT-AIRLOCK]	$a[\langle\, k(v) \,\rangle \uplus \Lambda] \to a.\,k(v)[\Lambda] \quad \big(k \notin \mathcal{K}(a)\big)$	
[OUT]	$a.\,k(v)[\Lambda] \to k(v), a[\Lambda] \quad \big(k \notin \mathcal{K}(a)\big)$	
[REACTION]	$\dfrac{\overrightarrow{\omega_1} \to \overrightarrow{\omega_2}}{\langle\, \overrightarrow{\omega_1[\theta]} \,\rangle \to \langle\, \overrightarrow{\omega_2[\theta]} \,\rangle}\ \big(\theta : \mathcal{X} \to \mathcal{V} \text{ valuation}\big)$	

Table 1. Chemical Abstract Machine – Generic Reduction Rules

ployment rules allow a collaboration to be decomposed until the agents stand

$$[\textsc{init}]_{\text{impure}} \qquad a.\,\sigma[\langle\,\rangle] \to a[\langle\,\sigma\,\rangle]$$

$$[\textsc{local}]_{\text{impure}} \qquad a \vdash \sigma_1, \overrightarrow{k_1(v_1)} \to \sigma_2, \overrightarrow{k_2(v_2)}$$

$$[\textsc{reaction}]_{\text{impure}} \quad \frac{a \vdash \overrightarrow{\lambda_1} \to \overrightarrow{\lambda_2}}{a \vdash \langle\, \overrightarrow{\lambda_1}[\theta]\,\rangle \to \langle\, \overrightarrow{\lambda_2}[\theta]\,\rangle}\ (\theta : \mathcal{X} \to \mathcal{V} \text{ valuation})$$

Table 2. Impure Criojo – Reduction Rules

in the web and become active with the creation of an empty local solution. The two communication rules [IN] allow a message $k(v)$ in transit to come through the airlock into the local solution associated to the agent providing the channel k. Symmetrically, the two communication rules [OUT] allow a message $k(v)$ in the local solution to go through the airlock into the web.

Besides the preceding rules, which are generic, there are rules specific to agents. Indeed, a program in Criojo defines not only a collaboration but also the behavior of the agents involved in the collaboration. In impure Criojo, the behavior of an agent is described by *any finite* presentation of a possibly *infinite* set of instances of the generic rule $[\textsc{local}]_{\text{impure}}$: see Table 2. These reduction rules can perform after the local solution has been initialized with the initial local state, as described by the rule $[\textsc{init}]_{\text{impure}}$. Associated to the specific rules, there is the law $[\textsc{reaction}]_{\text{impure}}$, allowing an instantiation: the free variables, which occur in the left hand side containing the premises are instantiated and then substituted in the right hand side containing the conclusions. Thus, impure Criojo is a language for wrappers: it allows an integration into a Criojo collaboration, by abstracting away from concrete devices for generating the rules specific to agents. We will see in the next section different examples. We now present pure Criojo: contrary to impure Criojo, it allows specific rules to be directly defined.

3.2 Introspection with Pure Criojo

A program in impure Criojo is not effective, in that it assumes some external device to generate a possibly infinite set of rules. In pure Criojo, the program is effective: there is a finite set of rules, which are given extensionally, so that no external device is needed. The machine is purely chemical.

Concretely, in pure Criojo, the state of an agent has a concrete representation, as an aggregate of internal messages. An internal message is an atom, a predicate applied to a value.

$$\text{Internal Messages: } R(v) \qquad R \in \mathcal{P} \text{ (Set of Predicates)}, v \in \mathcal{V} \text{ (Set of Values)}$$
$$\text{State: } \sigma ::= \emptyset \mid R(v)\,\&\,\sigma$$

Internal messages look like external ones. However, there is a difference between predicates and channels: contrary to channels, predicates are not terms

and therefore cannot occur in a value. In the following, we will follow a naming convention: channels will have a name with the first letter lowercase while predicates will have a name with the first letter capitalized. With the formal homogeneity between the internal messages defining the local state and the external messages, the local solution can become a multiset of messages, which is performed by the following initialization rule.

$$[\textsc{init}]_{\text{pure}} \quad a.\,(R(u)\,\&\,\sigma)[\Lambda] \to a.\,\sigma[\langle\, R(u)\,\rangle \uplus \Lambda] \qquad a.\,\emptyset[\Lambda] \to a[\Lambda]$$

Pure Criojo is defined as a language that allows syntactically describing the reductions of a local solution, defined as a multiset of messages, and semantically defining all the possible transformations of a local solution. This universality property requires an extension of standard chams. Indeed we now show a computability limitation for standard chams, where all reactions are assumed to be local with respect to an agent a, that is to say to conform to the following rules.

$$a \vdash \overrightarrow{\lambda_1} \to \overrightarrow{\lambda_2} \qquad \frac{a \vdash \overrightarrow{\lambda_1} \to \overrightarrow{\lambda_2}}{a \vdash \langle\, \overrightarrow{\lambda_1[\theta]}\,\rangle \to \langle\, \overrightarrow{\lambda_2[\theta]}\,\rangle} \qquad \frac{a \vdash \Lambda_1 \to \Lambda_2}{a \vdash \Lambda_1 \uplus \Lambda \to \Lambda_2 \uplus \Lambda}$$

The first rule expresses the local reduction of some messages (internal and external), and is a schema for the rules specific to the local cham associated to the agent. The first inference rule, an instance of the law [REACTION], allows the instantiation of the specific rules. The second inference rule, an instance of the law [CHEMICAL], allows the reduction to occur in any local solution. To express the computability limitation for standard chams, we need some terminology. Without loss of generality, we only consider internal messages. Predicates are arbitrarily split into two classes, the class of public predicates and the class of private ones. A *transformation* is a binary relation over multisets of messages, defined over public predicates. A transformation T is *computable* by a standard cham if there are (i) a finite set of local rules ($a \vdash \overrightarrow{\lambda_1} \to \overrightarrow{\lambda_2}$) and (ii) a multiset Λ_{i} of initial messages defined over private predicates such that for all multiset Λ_{in} in the input domain of the transformation T, we have: (i) for all multiset Λ_{out} associated to Λ_{in} by T, there exists an execution starting from the solution $\Lambda_{\text{i}} \uplus \Lambda_{\text{in}}$ and terminating with the solution $\Lambda_{\text{f}} \uplus \Lambda_{\text{out}}$, where Λ_{f} is some multiset of final messages defined over private predicates, and (ii) all execution starting from the solution $\Lambda_{\text{i}} \uplus \Lambda_{\text{in}}$ terminates, with a final solution $\Lambda_{\text{f}} \uplus \Lambda_{\text{out}}$, where Λ_{f} is some multiset of final messages defined over private predicates, and where $(\Lambda_{\text{in}}, \Lambda_{\text{out}})$ belongs to T. We can now formally specify the following limitation: a standard cham cannot compute a cloning transformation.

Proposition 1 (Clone Problem). *Given a public predicate R with arity zero, the transformation T equal to $(\langle\, R^n \,\rangle, \langle\, R^{2n} \,\rangle)_{n\in\mathbb{N}}$ cannot be computed by a standard cham (R^p means p occurrences of R).*

Proof. Suppose for a contradiction that there exists a standard cham computing transformation T. Let n be a natural number. There exists an execution starting from $\Lambda_{\text{i}} \uplus \langle\, R^n \,\rangle$ and terminating with $\Lambda_{\text{f}} \uplus \langle\, R^{2n} \,\rangle$ Then by applying the chemical law, we deduce an execution starting from $\Lambda_{\text{i}} \uplus \langle\, R^{n+1} \,\rangle$ and reaching

$\Lambda_\mathrm{f} \uplus \langle\, R^{2n+1} \,\rangle$. The last solution cannot be final. Hence there exists a rule that can be fired. If the rule consumes less than $2n+1$ messages R, then $\Lambda_\mathrm{f} \uplus \langle\, R^{2n} \,\rangle$ cannot be a final solution, contradiction. Therefore, for each n, there exists a rule that consumes $2n+1$ messages R. This is a contradiction, since the cham has a finite set of rules. □

To tackle this limitation, we extend the cham with introspection capacities. Thus, in pure Criojo, the behavior of an agent is described by a finite set of guarded rules $a \vdash \overrightarrow{\lambda_1} \to g \,?\, \overrightarrow{\lambda_2}$. Such a rule is fired only when the guard condition g is satisfied in the local solution formed with the internal messages, then consuming the premises $\overrightarrow{\lambda_1}$ and producing the conclusions $\overrightarrow{\lambda_2}$. External messages, which can freely come in or go out, are not considered in a guard to avoid race conditions. How to express guards? As we will see in the solution of the clone problem as well as in the following section, introspection is powerful when it allows to determine whether some rules are blocked. Hence guards are expressed in first-order logic, since a rule $a \vdash \overrightarrow{\lambda_1} \to g \,?\, \overrightarrow{\lambda_2}$ is blocked when the local solution satisfies $\neg(\exists\, \overrightarrow{x} \,.\, \langle\, \overrightarrow{\lambda_1} \,\rangle \wedge g)$, where $\overrightarrow{x}$ denotes the free variables in $\overrightarrow{\lambda_1}$ and $\langle\, \overrightarrow{\lambda_1} \,\rangle$ means that this multiset is included in the local solution.

Guard	g	$::= \langle\, \overrightarrow{R(u)} \,\rangle$	Presence of Local Messages
		$\mid \texttt{True} \mid \texttt{False} \mid g \vee g \mid g \wedge g \mid \neg g$	Propositional Guards
		$\mid \exists\, x \,.\, g \mid \forall\, x \,.\, g$	First-Order Guards

Semantically, we define the satisfaction judgment $\Lambda \vDash_\theta g$, which means that $g[\theta]$ is satisfied in the solution Λ. The satisfaction relation is defined as usual, following Tarski's interpretation over the set $\mathcal{V}$ of values, with one notable exception for atomic formulas.

$$
\begin{aligned}
\Lambda \vDash_\theta \langle\, \overrightarrow{R(u)} \,\rangle &\overset{\text{def}}{\Leftrightarrow} \langle\, \overrightarrow{R(u)}[\theta] \,\rangle \sqsubseteq \Lambda \\
\Lambda \vDash_\theta \texttt{True} &\overset{\text{def}}{\Leftrightarrow} \texttt{True} \\
\Lambda \vDash_\theta g_1 \vee g_2 &\overset{\text{def}}{\Leftrightarrow} \Lambda \vDash_\theta g_1 \vee \Lambda \vDash_\theta g_2 \\
\Lambda \vDash_\theta \neg g &\overset{\text{def}}{\Leftrightarrow} \neg(\Lambda \vDash_\theta g) \\
\Lambda \vDash_\theta \exists\, x \,.\, g &\overset{\text{def}}{\Leftrightarrow} \exists\, v \in \mathcal{V} \,.\, \Lambda \vDash_{\theta \oplus (v/x)} g
\end{aligned}
$$

The satisfaction relation is decidable, in time polynomial in the size of the solution Λ, as shown by Dantsin and Voronkov [8]. Finally, the cham for pure Criojo is defined as follows.

$$
[\textsc{local}]_\mathrm{pure} \qquad a \vdash \overrightarrow{R_1(u_1)}, \overrightarrow{k_1(v_1)} \to g \,?\, \overrightarrow{R_2(u_2)}, \overrightarrow{k_2(v_2)}
$$

$$
[\textsc{introspection}]_\mathrm{pure} \; \frac{a \vdash \overrightarrow{\lambda_1} \to g \,?\, \overrightarrow{\lambda_2} \quad \langle\, \overrightarrow{R_1(u_1)}[\theta] \,\rangle \uplus \Lambda \vDash_\theta g}{a \vdash \langle\, \overrightarrow{\lambda_1}[\theta] \,\rangle \uplus \Lambda \to \langle\, \overrightarrow{\lambda_2}[\theta] \,\rangle \uplus \Lambda} \left(\begin{array}{l} \overrightarrow{\lambda_1} = \overrightarrow{R_1(u_1)}, \overrightarrow{k_1(v_1)} \\ \overrightarrow{\lambda_2} = \overrightarrow{R_2(u_2)}, \overrightarrow{k_2(v_2)} \\ \Lambda = \langle\, \overrightarrow{R(u)} \,\rangle \\ \theta : \mathcal{X} \to \mathcal{V} \text{ valuation} \end{array} \right)
$$

The introspection rule combines the reaction rule expressing instantiation and the chemical rule expressing locality while adding the satisfaction of the guard. Note also that a guarded rule $a \vdash \overrightarrow{\lambda_1} \to \texttt{True} \,?\, \overrightarrow{\lambda_2}$ is equivalent to the standard rule $a \vdash \overrightarrow{\lambda_1} \to \overrightarrow{\lambda_2}$, so that we will often omit the guard `True`. With introspection, we can now solve the clone problem.

Proposition 2 (Clone Problem Revisited). *Given a public predicate R with arity zero, the transformation T equal to $(\langle\, R^n \,\rangle, \langle\, R^{2n} \,\rangle)_{n\in\mathbb{N}}$ can be computed by an introspective cham.*

Proof. Consider the following program in pure Criojo.

$$\texttt{One}, R \to \texttt{One}, S, S \qquad \texttt{One} \to \neg \langle\, R \,\rangle\, ?\, \texttt{Two} \qquad \texttt{Two}, S \to \texttt{Two}, R \qquad \texttt{Two} \to \neg \langle\, S \,\rangle\, ?\, \texttt{Three}$$

Then its execution starting from the local solution $\langle\, \texttt{One}, R^n \,\rangle$ terminates with the solution $\langle\, \texttt{Three}, R^{2n} \,\rangle$, for any n. □

3.3 Other Examples: Equality and Substitution

The translations into pure Criojo described in the next section require realizing some substitutions. It turns out that realizing a substitution also requires testing equality (between variables). We now give two programs written in pure Criojo to test term equality and to implement substitutions respectively. They are modular, since they are associated to two agents, `EQ`, a generic agent for equality and $\texttt{SUB}_\text{p}$, an agent for substitutions depending on an underlying nominal algebra, an algebra with binders.

Here is the program to test equality. It is generic, being independent from the signature for terms, thanks to introspection and instantiation.

$$\begin{aligned}
&\texttt{EQ} \vdash \texttt{isEqual}(v, u, k^+, k^-), \texttt{Session}(n) \to \texttt{RepEq}(n, k^+, k^-), \texttt{Eq}(n, v, u), \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{Session}(\texttt{succ}(n)) \\
&\texttt{EQ} \vdash \qquad \texttt{Eq}(n, v, u), \texttt{RepEq}(n, k^+, k^-) \to \neg \langle\, \texttt{Eq}(n, v, v) \,\rangle\, ?\, k^-(v, u) \\
&\texttt{EQ} \vdash \qquad \texttt{Eq}(n, v, v), \texttt{RepEq}(n, k^+, k^-) \to k^+(v)
\end{aligned}$$

To test the equality between v and u, a client sends the request $\texttt{isEqual}(v, u, k^+, k^-)$: channel `isEqual` is provided by agent `EQ`, and channels k^+ and k^-, provided by the client, are response channels used for positive tests and negative tests respectively. When the agent `EQ` receives a request, first it gets a new session identifier from the name server: thus the agent `EQ` can organize the responses to requests in parallel, each response being identified by the session identifier. The name server is implemented with the predicate `Session`, the constant 0 used to initialize the state with $\texttt{Session}(0)$ and the successor function `succ` used to increment the identifier n. Second the agent `EQ` generates two internal messages depending on the session identifier, one for the pending request, the other for testing equality. If the values v and u are distinct, the second rule is executed, which sends the negative response over k^-. Otherwise, the last rule is executed, which sends the positive response over k^+.

For substitutions, we do not consider universal algebras but nominal algebras [36], which are universal algebras with a built-in support for names and binding. This generalization is not only useful since the λ-calculus and the π-calculus, translated into Criojo, have binders, but also quite simple, since we only substitute closed terms to variables. Given a nominal algebra, we can define an agent $\texttt{SUB}_\text{p}$ computing substitutions over the algebra. The agent provides

a channel doSub, manipulated following a request-response protocol, as already seen for equality.

$$\mathtt{SUB_p} \vdash \mathtt{doSub}(t, v, a, k), \mathtt{Session}(n) \to \mathtt{RepSub}(n, k), \mathtt{Sub}(n, t, v, a), \mathtt{Session}(\mathtt{succ}(n))$$
$$\mathtt{SUB_p} \vdash \mathtt{RepSub}(n, k), \mathtt{ResSub}(n, t) \to k(t)$$

When the agent receives the message $\mathtt{doSub}(t, v, a, k)$, it computes the substitution $t[v/a]$, from $\mathtt{Sub}(n, t, v, a)$ to $\mathtt{ResSub}(n, t')$, and finally sends the result t' over k. A variant allows the client to send not only the channel but also a session identifier s used by the client to correlate the response with the calling computation. Thus, the response would become $k(s, v)$ instead of $k(v)$, the identifier s allowing correlation on the client side. We now detail the computation of substitutions in a nominal algebra.

A nominal signature has two basic sorts, ν for atoms (representing bound variables) and δ for data, and a set of functions f, each of which has an arity of the form $\tau \to \delta$. The left sort τ ranges over the sorts generated by the following grammar.

$$\text{Sort} \qquad \tau ::= 1 \mid \nu \mid \delta \mid \tau \times \tau \mid [\nu]\tau$$

A sort is either the singleton 1, used to represent the absence of arguments, the basic sorts ν and δ, used to represent a unique argument, an atom or a data, the Cartesian product $\tau \times \tau$, used to represent multiple arguments, and the non-standard sort $[\nu]\tau$, used to represent an argument that is an abstraction binding atoms. The corresponding terms are defined as follows.

$$\text{Term} \qquad t ::= () \mid a \mid f\, t \mid (t, t) \mid a.t$$

The term $a.t$, with sort $[\nu]\tau$, binds atom a in t. For instance, the λ-calculus can be described with the following signature.

$$\begin{aligned} \mathtt{var} &: \quad \nu \to \delta \\ \mathtt{app} &: \delta \times \delta \to \delta \\ \mathtt{lambda} &: \quad [\nu]\delta \to \delta \end{aligned}$$

We now recursively define the computations for substituting atom a with v in a term. The case for () is trivial.

$$\mathtt{SUB_p} \vdash \mathtt{Sub}(n, (), v, a) \to \mathtt{ResSub}(n, ())$$

For an atom b, we need to test equality, which is performed by calling agent EQ.

$$\begin{aligned} \mathtt{SUB_p} \vdash \quad & \mathtt{Sub}(n, b, v, a) \to \mathtt{isEqual}(b, a, \mathtt{equal}^+, \mathtt{equal}^-), \mathtt{WSub}(n, b, v, a) \\ \mathtt{SUB_p} \vdash \quad & \mathtt{equal}^+(a), \mathtt{WSub}(n, a, v, a) \to \mathtt{ResSub}(n, v) \\ \mathtt{SUB_p} \vdash \quad & \mathtt{equal}^-(b, a), \mathtt{WSub}(n, b, v, a) \to \mathtt{ResSub}(n, b) \end{aligned}$$

We now deal with the recursive cases. There are two phases: a top-down phase that makes the recursive calls, and a bottom-up phase that synthesize the result. Here is the top-down phase. First, for a function or a pair, one or two recursive calls are generated. Each recursive call has an identifier, provided by the name server RecCall. The internal messages $\mathtt{Op}(n, f, m)$ and $\mathtt{Pair}(n, m, \mathtt{succ}(m))$ give

the links between the different identifiers, assigned to the caller and to the callees, which later will allow the result to be synthesized.

$$\mathtt{SUB_p} \vdash \mathtt{Sub}(n, f\,t, v, a), \mathtt{RecCall}(m) \to \mathtt{Sub}(m, t, v, a), \mathtt{Op}(n, f, m), \mathtt{RecCall}(\mathtt{succ}(m))$$

$$\mathtt{SUB_p} \vdash \mathtt{Sub}(n, (t_1, t_2), v, a), \mathtt{RecCall}(m) \to \mathtt{Sub}(m, t_1, v, a), \mathtt{Sub}(\mathtt{succ}(m), t_2, v, a), \mathtt{Pair}(n, m, \mathtt{succ}(m)), \mathtt{RecCall}(\mathtt{succ}(\mathtt{succ}(m)))$$

Second, for a binding abstraction, an equality test is also needed: only free atoms are substituted.

$$\mathtt{SUB_p} \vdash \mathtt{Sub}(n, b.t, v, a) \to \mathtt{isEqual}(b, a, \mathtt{equal}^+, \mathtt{equal}^-), \mathtt{WSub}(n, b.t, v, a)$$

$$\mathtt{SUB_p} \vdash \mathtt{equal}^+(a), \mathtt{WSub}(n, a.t, v, a) \to \mathtt{ResSub}(n, a.t)$$

$$\mathtt{SUB_p} \vdash \mathtt{equal}^-(b, a), \mathtt{WSub}(n, b.t, v, a), \mathtt{RecCall}(m) \to \mathtt{Sub}(m, t, v, a), \mathtt{Binder}(n, b, m), \mathtt{RecCall}(\mathtt{succ}(m))$$

Now we come to the bottom-up phase: the results are collected at some level and then synthesized at the upper level, thanks to the internal messages $\mathtt{Op}(n, f, m)$, $\mathtt{Pair}(n, m_1, m_2)$ and $\mathtt{Binder}(n, a, m)$, which keep the links between recursive calls.

$$\mathtt{SUB_p} \vdash \mathtt{Op}(n, f, m), \mathtt{ResSub}(m, v) \to \mathtt{ResSub}(n, f\,v)$$

$$\mathtt{SUB_p} \vdash \mathtt{Pair}(n, m_1, m_2), \mathtt{ResSub}(m_1, v_1), \mathtt{ResSub}(m_2, v_2) \to \mathtt{ResSub}(n, (v_1, v_2))$$

$$\mathtt{SUB_p} \vdash \mathtt{Binder}(n, a, m), \mathtt{ResSub}(m, v) \to \mathtt{ResSub}(n, a.v)$$

How can we prove that this implementation is right? We propose a systematic method to answer questions like this one, described below for the case of substitutions. First, derive from the standard definition of substitutions with an inference system an implementation in impure Criojo, for a new agent $\mathtt{SUB_i}$. The reduction relation is generated by an inference rule that looks like the following one, giving a benchmark definition.

$$\frac{t[v/a] \mapsto t'}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, t, v, a) \to \mathtt{ResSub}(n, t')}$$

Second, we prove that the pure agent $\mathtt{SUB_p}$ and the impure agent $\mathtt{SUB_i}$ are equivalent. We now formalize this notion of equivalence.

3.4 Bisimilarity

Consider a web Ω and a subset A of its agents. The projection of Ω over A is the web including in Ω and containing all the entities associated to agents in A:

- active agents $a[\Lambda]$ or $a.\,\alpha[\Lambda]$, with a in A,
- messages $k(v)$ in transit, with k provided by some agent in A.

In the following, a projection over A is also called a collaboration over A, since it represents the semantic counterpart of a syntactic collaboration. Given a collaboration Ω over A, we denote by $\mathcal{K}(\Omega)$ the set of the channels provided by the agents in A. We deal with the following question: can we replace a collaboration

with another one observationally equivalent, that is to say such that the environment can never observe any difference? Following the Black Box principle, the environment interacts with a collaboration only by exchanging messages. First difficulty: when we replace a collaboration by another one providing extra channels, the environment can trivially observe a difference, the ability to communicate over the extra channels. Thus, we should restrict ourselves to collaborations providing exactly the same set of channels. But it is not a good idea: it would prevent from decomposing an agent into multiple agents to provide the same services. Therefore we may need a firewall between a collaboration and its environment. It is represented as a restriction operator, reminiscent of the language CCS: given a collaboration Ω and a subset K of $\mathcal{K}(\Omega)$, $\Omega \setminus K$ represents a collaboration where the communications with the environment over the channels in K are forbidden. Second difficulty: the operational semantics, defined by a reduction relation, does not account for the interactions with an environment. Thus, to formalize observational equivalence, we turn the reduction relation into a labeled transition system, following a standard technique [2]. We consider as actions the silent action, τ, input messages, denoted $+k(v)$, and output messages, denoted $-k(v)$. For collaborations $\Omega \setminus K$ and $\Omega' \setminus K$, denoting by $\rightarrow^*$ the reflexive and transitive closure of the reduction relation $\rightarrow$,

- we write $(\Omega \setminus K) \stackrel{\tau}{\Longrightarrow} (\Omega' \setminus K)$ if $\Omega \rightarrow^* \Omega'$;
- we write $(\Omega \setminus K) \stackrel{+k(v)}{\Longrightarrow} (\Omega' \setminus K)$ if there exists a collaboration Ω_1 and an active agent $a[\Lambda]$ such that

$$\Omega \rightarrow^* \Omega_1 \uplus \langle\, a[\Lambda] \,\rangle, \qquad \Omega_1 \uplus \langle\, a.\, k(v)[\Lambda] \,\rangle \rightarrow^* \Omega' \qquad \text{and} \quad k \in \mathcal{K}(a) - K;$$

- we write $(\Omega \setminus K) \stackrel{-k(v)}{\Longrightarrow} (\Omega' \setminus K)$ if there exists a collaboration Ω_1 and an active agent $a[\Lambda]$ such that

$$\Omega \rightarrow^* \Omega_1 \uplus \langle\, a.\, k(v)[\Lambda] \,\rangle, \qquad \Omega_1 \uplus \langle\, a[\Lambda] \,\rangle \rightarrow^* \Omega' \qquad \text{and} \quad k \notin \mathcal{K}(\Omega).$$

For any labeled transition system, there is a standard notion of bisimulation.

Definition 1 (Simulation – Bisimulation – Bisimilarity). *Let R be a relation over collaborations. R is a* simulation *if for any ordered pair $(\Omega_1 \setminus K_1, \Omega_2 \setminus K_2)$ in R, whenever $(\Omega_1 \setminus K_1) \stackrel{X}{\Longrightarrow} (\Omega_1' \setminus K_1)$, there exists Ω_2' such that $(\Omega_2 \setminus K_2) \stackrel{X}{\Longrightarrow} (\Omega_2' \setminus K_2)$ and $(\Omega_1' \setminus K_1, \Omega_2' \setminus K_2) \in R$. R is a* bisimulation *if R and R^{-1} are simulations.*

Two collaborations $\Omega_1 \setminus K_1$ and $\Omega_2 \setminus K_2$, are bisimilar *if there exists a bisimulation containing $(\Omega_1 \setminus K_1, \Omega_2 \setminus K_2)$.*

We do not develop further the theory of bisimulation because the previous definition is enough to state the main properties of the translations in the next section and to state the equivalence between both versions for substitutions, as shown below. The development of the theory probably requires extending the grammar for syntactic collaborations with the restriction operator and the chemical abstract machine with firewalls: we let this extension to a future work.

Coming back to substitutions, we can now prove the equivalence between both definitions, the impure one and the pure one.

We first define the impure version for substitutions. The state of the impure agent $\mathtt{SUB_i}$ is represented as an aggregate combining (i) a global counter

$$\mathtt{Session}(n)$$

for the identifiers of the internal sessions, and (ii) pending requests. A pending request is a sub-aggregate, either

$$\mathtt{Sub}(n, t, v, a) \,\&\, \mathtt{RepSub}(n, k)$$

or

$$\mathtt{ResSub}(n, t') \,\&\, \mathtt{RepSub}(n, k).$$

It indicates that in the internal session n, the agent either has to evaluate $t[v/a]$ or has evaluated $t[v/a]$ into t', and will respond over channel k. The join operator & is assumed to be associative and commutative, which leads to the following rules.

$$\sigma_1 \,\&(\sigma_2 \,\&\, \sigma_3) \equiv (\sigma_1 \,\&\, \sigma_2) \,\&\, \sigma_3 \qquad \sigma_1 \,\&\, \sigma_2 \equiv \sigma_2 \,\&\, \sigma_1$$

$$\frac{\sigma_1 \equiv \sigma_1' \quad \mathtt{SUB_i} \vdash \sigma_1' \to \sigma_2' \quad \sigma_2' \equiv \sigma_2}{\mathtt{SUB_i} \vdash \sigma_1 \to \sigma_2}$$

Like $\mathtt{SUB_p}$, the impure agent $\mathtt{SUB_i}$ provides a channel doSub, manipulated following the same request-response protocol.

$$\mathtt{SUB_i} \vdash \mathtt{doSub}(t, v, a, k), \sigma \,\&\, \mathtt{Session}(n) \to \mathtt{RepSub}(n, k) \,\&\, \mathtt{Sub}(n, t, v, a) \,\&\, \sigma \,\&\, \mathtt{Session}(\mathtt{succ}(n))$$

$$\mathtt{SUB_i} \vdash \mathtt{RepSub}(n, k) \,\&\, \mathtt{ResSub}(n, t) \,\&\, \sigma \to \sigma, k(t)$$

The reduction rules $\mathtt{SUB_i} \vdash \mathtt{Sub}(n, t, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, t') \,\&\, \sigma$ are defined in Table 3 by a syntax-directed inference system, directly adapted from the standard definition.

We finally show the equivalence between the pure agent $\mathtt{SUB_p}$ and the impure one $\mathtt{SUB_i}$.

Theorem 1 (Substitutions – Bisimilarity). *Let K be the following set of channels:*

$$\{\mathtt{isEqual}, \mathtt{equal}^+, \mathtt{equal}^-\}.$$

Then the collaborations

$$\langle\, \mathtt{SUB_i}[\langle\, \mathtt{Session}(0) \,\rangle] \,\rangle \setminus \emptyset$$

and

$$\langle\, \mathtt{SUB_p}[\langle\, \mathtt{Session}(0), \mathtt{RecCall}(0) \,\rangle], \mathtt{EQ}[\langle\, \mathtt{Session}(0) \,\rangle] \,\rangle \setminus K$$

are bisimilar.

Proof. We sketch the proof.

$$\frac{}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, (), v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, ()) \,\&\, \sigma}$$

$$\frac{}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, b, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, v) \,\&\, \sigma} \quad (b = a)$$

$$\frac{}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, b, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, b) \,\&\, \sigma} \quad (b \neq a)$$

$$\frac{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, t, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, t') \,\&\, \sigma}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, f\,t, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, f\,t') \,\&\, \sigma}$$

$$\frac{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, t_1, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, t_1') \,\&\, \sigma \quad \mathtt{SUB_i} \vdash \mathtt{Sub}(n, t_2, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, t_2') \,\&\, \sigma}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, (t_1, t_2), v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, (t_1', t_2')) \,\&\, \sigma}$$

$$\frac{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, t, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, t') \,\&\, \sigma}{\mathtt{SUB_i} \vdash \mathtt{Sub}(n, b.t, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, b.t') \,\&\, \sigma} \quad (b \neq a)$$

$$\mathtt{SUB_i} \vdash \mathtt{Sub}(n, b.t, v, a) \,\&\, \sigma \to \mathtt{ResSub}(n, b.t) \,\&\, \sigma \quad (b = a)$$

Table 3. Substitutions in Impure Criojo

We split the relations and channels of $\mathtt{SUB_i}$ and $\mathtt{SUB_p}$ into three sets:

$$\begin{aligned} H &= \{\mathtt{doSub}\ \text{(request)}, k\ \text{(response)}\}, \\ I &= \{\mathtt{Session}\ \text{(for substitutions)}, \mathtt{RepSub}, \mathtt{ResSub}\}, \\ J &= \{\mathtt{Sub}, \mathtt{WSub}, \mathtt{Op}, \mathtt{Pair}, \mathtt{Binder}, \mathtt{RecCall}, \mathtt{equal}^+, \mathtt{equal}^-, \mathtt{isEqual}, \\ &\quad \ldots \text{(equality relations)}\}. \end{aligned}$$

We exhibit a bisimulation R containing both collaborations. Consider collaborations

$$\langle\, \mathtt{SUB_i}[\langle\, A_1 \,\&\, C_1 \,\rangle \uplus B_1] \,\rangle \setminus \emptyset$$

and

$$(\langle\, \mathtt{SUB_p}[A_2 \uplus C_2 \uplus B_2], \mathtt{EQ}[D_2] \,\rangle \uplus E_2) \setminus K$$

where

- B_1 and B_2 contain the messages over H,
- A_1 and A_2 contain the messages over I,
- C_1, C_2, D_2 and E_2 contain the messages over J.

They are related by R if the following conditions are satisfied.

(A) There is a biunivocal correspondence between the impure state A_1, B_1 and the pure state A_2, B_2, built as follows.

$$\begin{aligned} \mathtt{doSub}(t, v, a, k) &\longleftrightarrow \mathtt{doSub}(t, v, a, k) \\ k(t) &\longleftrightarrow k(t) \\ \mathtt{Session}(n) &\longleftrightarrow \mathtt{Session}(n) \\ \mathtt{RepSub}(n, k) &\longleftrightarrow \mathtt{RepSub}(n, k) \\ \mathtt{ResSub}(n, t) &\longleftrightarrow \mathtt{ResSub}(n, t) \end{aligned}$$

(B) We denote by X the set of session identifiers occurring in C_1:

$$C_1 = \langle\, \mathtt{Sub}(n, t_n, v_n, a_n) \mid n \in X \,\rangle$$

If $n \in X$, whereas there is an internal message $\mathtt{Sub}(n, t_n, v_n, a_n)$ in the impure state C_1, either there is a corresponding message in the pure state C_2, or there is no corresponding message, the pure agent being computing the substitution $t_n[v_n/a_n]$. Therefore, we add the following condition: the collaboration

$$\langle\, \mathtt{SUB}_{\mathrm{p}}[C_2], \mathtt{EQ}[D_2] \,\rangle \uplus E_2$$

converges to the following final state $\langle\, \mathtt{ResSub}(n, t'_n) \mid n \in X \,\rangle$, such that for any n, we have $t'_n = t_n[v_n/a_n]$.

Finally, it remains to prove that R is a bisimulation. We need to consider three cases, corresponding to the silent action, an input message (over channel `doSub`) and an output message (over channels k). We deduce the simulation properties from the following lemmas.

(1) First Lemma – Computational Correction
Assume the collaboration

$$\langle\, \mathtt{SUB}_{\mathrm{p}}[C_2], \mathtt{EQ}[D_2] \,\rangle \uplus E_2$$

converges to the following final state

$$\langle\, \mathtt{ResSub}(n, t'_n) \mid n \in X \,\rangle\,.$$

Then given a fresh identifier m, the collaboration

$$\langle\, \mathtt{SUB}_{\mathrm{p}}[C_2 \uplus \langle\, \mathtt{Sub}(m, t, v, a) \,\rangle], \mathtt{EQ}[D_2] \,\rangle \uplus E_2)$$

converges to the following final state

$$\langle\, \mathtt{ResSub}(n, t'_n) \mid n \in X \,\rangle \uplus \langle\, \mathtt{ResSub}(m, t') \,\rangle,$$

such that $t' = t[v/a]$. By induction over term t.

(2) Second Lemma – Operational Simulation
Starting from two collaborations

$$\Lambda_1 \stackrel{\mathtt{def}}{=} \langle\, \mathtt{SUB}_{\mathrm{i}}[\langle\, A_1 \,\&\, C_1 \,\rangle \uplus B_1] \,\rangle \text{ and } \Lambda_2 \stackrel{\mathtt{def}}{=} \langle\, \mathtt{SUB}_{\mathrm{p}}[A_2 \uplus C_2 \uplus B_2], \mathtt{EQ}[D_2] \,\rangle \uplus E_2$$

such that $(\Lambda_1 \setminus \emptyset, \Lambda_2 \setminus K)$ belongs to R, if $\Lambda_1 \to \Lambda'_1$, then there exists Λ'_2 such that $\Lambda_2 \to^* \Lambda'_2$ and $(\Lambda'_1 \setminus \emptyset, \Lambda'_2 \setminus K)$ belongs to R, and conversely if $\Lambda_2 \to \Lambda'_2$, then there exists Λ'_1 such that $\Lambda_1 \to^* \Lambda'_1$ and $(\Lambda'_1 \setminus \emptyset, \Lambda'_2 \setminus K)$ belongs to R. By a case analysis over reductions and by using the preceding lemma.

□

Finally, to conclude the section, we examine to what extent the language Criojo meets the requirements. Pure Criojo, with its introspective capacities, is a candidate for a universal language for compiling orchestration languages, whereas impure Criojo, with its capacities for wrapping, is a candidate for a universal language for interfacing resources. The next section essentially aims at

arguing these claims. The chemical abstract machine that we have designed for service-oriented computing is clearly based on a message-passing model: molecules represent messages and some obvious chemical rules account for message communication. Channel mobility is also an important feature of our cham: it allows complex protocols to be implemented in a simple way, by passing channels as values. We have not dealt with scope management explicitly: as seen through the previous examples, it essentially corresponds to a specific discipline in the management of names like session identifiers by using name servers. The chemical model, as developed, also adheres to the Black Box principle. The bisimulation theory that we have sketched formalizes the principle and will lead to applications in the next section.

4 Translation of Four Idiomatic Languages

We now give the formal translations of the four idiomatic languages that we have chosen: a variant of Dijkstra's language of guarded commands, Datalog with negation, a λ-calculus extended and the π-calculus. Actually, except for Datalog, where there is a computation but no communication, we proceed as described at the end of the preceding section: in a distributed context, we give two versions, the former in impure Criojo, considered as the benchmark definition, the latter in pure Criojo, and then we prove that they are equivalent by giving a bisimilarity result.

4.1 Dijkstra's Language of Guarded Commands

As shown by the solution to the clone problem, sequencing can be managed by tokens. Instead of this low-level management, we would rather writing the following program:

$$\texttt{do}\,\{R \multimap g\,?\,S, S\}\,;\,\texttt{do}\,\{S \multimap g\,?\,R\}.$$

The command `do`, corresponding to a loop, allows the rule to be repeated until blocking, which entails its termination. Note the new notation for a rule, reminiscent of Linear Logic: indeed, a rule is now consumed after it has been fired. Commands can also be sequenced. If we add a blocking alternative, we get the following variant of Dijkstra's language of guarded commands.

Script	$s ::= \texttt{skip} \mid s\,;\,s \mid \texttt{if}\,\{c\} \mid \texttt{do}\,\{c\}$
Guarded Command Set	$c ::= r \triangleright s \mid c \parallel c$
Guard Rule	$r ::= M \multimap g\,?\,M$
Messages	$M ::= \overrightarrow{R(u)}, \overrightarrow{k(v)}$

Compared with Dijkstra's language, there are two differences, related: first, there is no atomic action, except the empty one `skip`, second, the guard of a command becomes a one-shot rule, with a side-effect, called a guard rule, thus compensating the lack of actions. In the following, a guard rule with no message ($\emptyset \multimap g\,?\,\emptyset$) is simply denoted by its guard (g). The empty action is also omitted, $r \triangleright \texttt{skip}$ becoming r.

The operational semantics of the language is described in Table 4. It is a variant of the one given by Plotkin for Dijkstra's language of guarded commands [30], following the Structural Operational Semantics style. It involves two relations, the first one to reduce a configuration composed of a script and a solution, and the second one to select a guarded command. More precisely, $(s, \Lambda) \Rightarrow (s', \Lambda')$ means that the script s with solution Λ reduces to the script s' with solution Λ', while $(s, \Lambda) \Rightarrow \Lambda$ means that the script s cannot reduce in solution Λ and terminates. Given a set of guarded commands c and a solution Λ, $(c, \Lambda) \mapsto (M, M', s)$ means that a guarded command has been selected in c, producing two multisets of messages, M to be removed, and M' to be added, and a script s to be executed, while $(c, \Lambda) \mapsto \bot$ means that the selection fails.

$$(\mathtt{skip}, \Lambda) \Rightarrow \Lambda \qquad \frac{(s_1, \Lambda) \Rightarrow (s_1', \Lambda')}{(s_1 \,;\, s_2, \Lambda) \Rightarrow (s_1' \,;\, s_2, \Lambda')} \qquad \frac{(s_1, \Lambda) \Rightarrow \Lambda}{(s_1 \,;\, s_2, \Lambda) \Rightarrow (s_2, \Lambda)}$$

$$\frac{(c, \Lambda) \mapsto (M^-, M^+, s)}{(\mathtt{if}\,\{c\}, \Lambda) \Rightarrow (s, \Lambda \uplus \langle\, M^+ \,\rangle \text{-} \langle\, M^- \,\rangle)}$$

$$\frac{(c, \Lambda) \mapsto (M^-, M^+, s)}{(\mathtt{do}\,\{c\}, \Lambda) \Rightarrow (s \,;\, \mathtt{do}\,\{c\}, \Lambda \uplus \langle\, M^+ \,\rangle \text{-} \langle\, M^- \,\rangle)} \qquad \frac{(c, \Lambda) \mapsto \bot}{(\mathtt{do}\,\{c\}, \Lambda) \Rightarrow \Lambda}$$

$$\frac{\Lambda \vDash_\theta M \wedge g}{((M \multimap g \,?\, M') \triangleright s, \Lambda) \mapsto (M[\theta], M'[\theta], s)} \qquad \frac{\Lambda \vDash \neg(\exists\, \overrightarrow{x} \,.\, M \wedge g)}{((M \multimap g \,?\, M') \triangleright s, \Lambda) \mapsto \bot}$$

$$\frac{(c_i, \Lambda) \mapsto (M, M', s)}{(c_1 \parallel c_2, \Lambda) \mapsto (M, M', s)}\ (i \in \{1, 2\}) \qquad \frac{(c_1, \Lambda) \mapsto \bot \quad (c_2, \Lambda) \mapsto \bot}{(c_1 \parallel c_2, \Lambda) \mapsto \bot}$$

Table 4. Guarded Commands – Small-Step Operational Semantics

Thanks to the impure version of Criojo, we can embed Dijkstra's language into Criojo in a straightforward way: it suffices to translate the operational semantics into rules for the cham. The state of an agent $\mathtt{GC_i}$ that implements Dijkstra's language is a pair $s \,\&\, \Lambda$ composed of a script s and a multiset of internal messages Λ. We add an extra state $\top \,\&\, \Lambda$ to indicate that the script has terminated its execution because it could not reduce. The rules of the cham are generated by the following inferences rules.

$$\frac{(s_1, \Lambda_1 \uplus \langle\, \overrightarrow{k_1(v_1)} \,\rangle) \Rightarrow (s_2, \Lambda_2 \uplus \langle\, \overrightarrow{k_2(v_2)} \,\rangle)}{\mathtt{GC_i} \vdash s_1 \,\&\, \Lambda_1, \overrightarrow{k_1(v_1)} \to s_2 \,\&\, \Lambda_2, \overrightarrow{k_2(v_2)}} \qquad \frac{(s, \Lambda \uplus \langle\, \overrightarrow{k(v)} \,\rangle) \Rightarrow \Lambda \uplus \langle\, \overrightarrow{k(v)} \,\rangle}{\mathtt{GC_i} \vdash s \,\&\, \Lambda \to \top \,\&\, \Lambda}$$

The rules of the agent $\mathtt{GC_i}$ are generic: they do not depend on the script to be executed, since the dependence comes from the injection of the script into the initial state.

Now, we translate into pure Criojo Dijkstra's language of guarded commands. The translation of each command depends on two tokens, B for "Begin" and E for "End", which are used to manage the scheduling of commands.

$$\begin{aligned}
\mathcal{D}(\texttt{skip})_{B,E} &= B \to \texttt{True}\,?\,E \\
\mathcal{D}(s_1\,;\,s_2)_{B,E} &= \nu\, I.\mathcal{D}(s_1)_{B,I}, \mathcal{D}(s_2)_{I,E} \\
\mathcal{D}(\texttt{if}\,\{c\})_{B,E} &= \mathcal{D}(c)_{B,E} \\
\mathcal{D}(\texttt{do}\,\{c\})_{B,E} &= \mathcal{D}(c)_{B,B}, (B \to \mathcal{G}(c)\,?\,E)
\end{aligned}$$

The empty script converts the begin token into the end token. The sequence $s_1\,;\,s_2$ requires an intermediate fresh token (cf. $\nu\, I.-$), which corresponds to the end of s_1 and the beginning of s_2. The translation of the alternative and the loop depends on the translation of the associated set of guarded commands. Note the differences: for the loop, the translation uses the same token, allowing a repetition, and adds a rule to quit the loop, when its guard rules cannot be fired. A guarded command is translated into a rule and the translation of the continuation script. Their sequencing results from the use of an intermediate fresh token.

$$\begin{aligned}
\mathcal{D}((M \multimap g\,?\,M') \triangleright s)_{B,E} &= \nu\, I.(M, B \to g\,?\,M', I), \mathcal{D}(s)_{I,E} \\
\mathcal{D}(c_1 \parallel c_2)_{B,E} &= \mathcal{D}(c_1)_{B,E}, \mathcal{D}(c_2)_{B,E}
\end{aligned}$$

Finally, given a set c of guarded commands, the guard $\mathcal{G}(c)$ expresses that the guard rules cannot be fired.

$$\begin{aligned}
\mathcal{G}((M \multimap g\,?\,M') \triangleright s) &= \neg(\exists\, \overrightarrow{x}\,.\, M \wedge g) \\
\mathcal{G}(c_1 \parallel c_2) &= \mathcal{G}(c_1) \wedge \mathcal{G}(c_2)
\end{aligned}$$

Note that we need to assume that the premise M only contains internal messages in order to get a guard defined over internal messages. This assumption is reasonable because it prevents race conditions from happening: since external messages can freely come in or go out, the termination of a loop ought not to depend on external messages.

Conversely, any program R in Criojo can be represented as a script in the language of guarded commands. First, the rules are translated into guarded commands. Second, a program is translated into a loop. To avoid the termination of the loop, the rules translated belong to an alternative, which can block and wait, and is guarded with `True`.

$$\begin{aligned}
\mathcal{E}_{\mathrm{R}}(M \to g\,?\,M') &= (M \multimap g\,?\,M') \triangleright \texttt{skip} \\
\mathcal{E}_{\mathrm{R}}(R_1, R_2) &= \mathcal{E}_{\mathrm{R}}(R_1) \parallel \mathcal{E}_{\mathrm{R}}(R_2) \\
\mathcal{E}_{\mathrm{P}}(R) &= \texttt{do}\,\{\texttt{True} \triangleright \texttt{if}\,\{\mathcal{E}_{\mathrm{R}}(R)\}\}
\end{aligned}$$

The variant $\mathcal{E}_{\mathrm{P}}(R) = \texttt{do}\,\{\mathcal{E}_{\mathrm{R}}(R)\}$, simpler, does not work: indeed, in a situation where no rule could be selected, the program would terminate.

We now study the equivalence between the scripts in Dijkstra's language and the programs in Criojo, for both translations. First, we deal with the translation into pure Criojo.

Theorem 2 (Guarded Commands to Pure Criojo – Bisimilarity). *Let s_0 be a script in Dijkstra's language of guarded commands, and let $\mathcal{D}(s_0)_{B,E}$ be its translation, for distinct tokens B and E. Consider the generic impure agent* $\texttt{GC}_{\mathrm{i}}$

implementing Dijkstra's language and the pure agent $\mathtt{GC}_\mathrm{p}$ *with rules* $\mathcal{D}(s_0)_{B,E}$. *Then for any multiset* Λ_in *of internal messages and any multiset* Λ_ex *of external messages, the collaborations*

$$\langle\, \mathtt{GC}_\mathrm{i}[\langle\, s_0 \,\&\, \Lambda_\mathrm{in} \,\rangle \uplus \Lambda_\mathrm{ex}] \,\rangle$$

and

$$\langle\, \mathtt{GC}_\mathrm{p}[\langle\, B \,\rangle \uplus \Lambda_\mathrm{in} \uplus \Lambda_\mathrm{ex}] \,\rangle$$

are bisimilar.

Proof. We sketch the proof.

First, we need some results about the translation of a script. Given a translation $\mathcal{D}(s)_{B,E}$ of a script s with respect to tokens B and E, we can define a transitive relation $\prec$ between the tokens used in the translation: it is the transitive closure of the relation that contains (I_1, I_2) if there exists in the translation some rule where token I_1 occurs on the left hand side and token I_2 occurs on the right hand side. Token B is a least element of $\prec$ while E is a greatest element of $\prec$: $\forall\, I\,.\,(I \neq B) \Rightarrow (B \prec I) \wedge (I \neq B) \Rightarrow (I \prec E)$. Moreover, if $B \neq E$, E is also a maximal element of $\prec$: $\forall\, I\,.\,\neg(E \prec I)$. Note that this is not true for B because of loops. We also define a notion of restriction for translations: given a token I, the script $\pi_I \mathcal{D}(s)_{B,E}$ contains all the rules of $\mathcal{D}(s)_{B,E}$ involving tokens I_1 and I_2 such that $I \preceq I_1$ and $I \preceq I_2$.

Second, we exhibit a bisimulation R containing both collaborations. Consider collaborations

$$\langle\, \mathtt{GC}_\mathrm{i}[\langle\, s \,\&\, \Lambda_\mathrm{in} \,\rangle \uplus \Lambda_\mathrm{ex}] \,\rangle$$

and

$$\langle\, \mathtt{GC}_\mathrm{p}[\langle\, I \,\rangle \uplus \Lambda_\mathrm{in} \uplus \Lambda_\mathrm{ex}] \,\rangle\,.$$

They are related by R if the following conditions are satisfied.

(A) If s is different from $\top$, then $\pi_I \mathcal{D}(s)_{B,E} = \mathcal{D}(s)_{I,E}$.

(B) If s is equal to $\top$, then $I = E$.

Finally, it remains to prove that R is a bisimulation. We deduce the simulation properties from the following lemmas.

(1) First Lemma – Restriction Composition

For all tokens I_1 and I_2 with $I_1 \prec I_2$, we have: $\pi_{I_2} \pi_{I_1} \mathcal{D}(s)_{B,E} = \pi_{I_2} \mathcal{D}(s)_{B,E}$.

(2) Second Lemma – Reduction Simulation

First, $(s, \Lambda) \Rightarrow (s', \Lambda')$ if and only if for any pair (B, E), there exists a token I different from B and E such that (i) the script $\mathcal{D}(s)_{B,E}$ entails reduction

$$\langle\, B \,\rangle \uplus \Lambda \rightarrow \langle\, I \,\rangle \uplus \Lambda'$$

and (ii) $\pi_I \mathcal{D}(s)_{B,E} = \mathcal{D}(s')_{I,E}$.

Second, $(s, \Lambda) \Rightarrow \Lambda$ if and only if the script $\mathcal{D}(s)_{B,E}$ entails reduction

$$\langle\, B \,\rangle \uplus \Lambda \rightarrow \langle\, E \,\rangle \uplus \Lambda.$$

□

Second, we deal with the translation from pure Criojo.

Theorem 3 (Pure Criojo to Guarded Commands – Bisimilarity). *Let R be a script in pure Criojo and let $\mathcal{E}_{\mathrm{P}}(R)$ be its translation into Dijkstra's language of guarded commands. Consider the pure agent $\mathtt{GC}_{\mathrm{p}}$ with rules R and the generic impure agent $\mathtt{GC}_{\mathrm{i}}$ implementing Dijkstra's language. Then for any multiset Λ_{in} of internal messages and any multiset Λ_{ex} of external messages, the collaborations*

$$\langle\, \mathtt{GC}_{\mathrm{p}}[\Lambda_{\mathrm{in}} \uplus \Lambda_{\mathrm{ex}}] \,\rangle$$

and

$$\langle\, \mathtt{GC}_{\mathrm{i}}[\langle\, \mathcal{E}_{\mathrm{P}}(R) \,\&\, \Lambda_{\mathrm{in}} \,\rangle \uplus \Lambda_{\mathrm{ex}}] \,\rangle$$

are bisimilar.

Proof. We sketch the proof.

We exhibit a bisimulation R containing both collaborations. Consider collaborations

$$\langle\, \mathtt{GC}_{\mathrm{p}}[\Lambda_{\mathrm{in}} \uplus \Lambda_{\mathrm{ex}}] \,\rangle$$

and

$$\langle\, \mathtt{GC}_{\mathrm{i}}[\langle\, s \,\&\, \Lambda_{\mathrm{in}} \,\rangle \uplus \Lambda_{\mathrm{ex}}] \,\rangle\,.$$

They are related by R if s is equal to either $\mathcal{E}_{\mathrm{P}}(R)$, $\mathtt{if}\,\{\mathcal{E}_{\mathrm{R}}(R)\}\,;\,\mathcal{E}_{\mathrm{P}}(R)$ or $\mathtt{skip}\,;\,\mathcal{E}_{\mathrm{P}}(R)$.

Finally, it remains to prove that R is a bisimulation, which is trivial. □

As defined, because Dijkstra's language is sequential, it is not well-suited to a distributed context. In a typical application, multiple clients request a sequential service: each request leads to a thread that concurrently executes with other threads. It is straightforward to encode a concurrent behavior like this with slight modifications to our translation. First, a request-response protocol is implemented, as already seen in the preceding section.

$$\begin{aligned}
&\mathtt{GC}_{\mathrm{p}} \vdash && \mathtt{Session}(n),\quad \mathtt{Session}(\mathtt{succ}(n)),\\
&&& \mathtt{l}(a,k) \to \mathtt{Begin}(n), \mathtt{Arg}(n,a), \mathtt{Rep}(n,k)\\
&\mathtt{GC}_{\mathrm{p}} \vdash && \mathtt{End}(n), \mathtt{Return}(n,r), \mathtt{Rep}(n,k) \to k(r)
\end{aligned}$$

The agent $\mathtt{GC}_{\mathrm{p}}$ provides a channel $\mathtt{l}$. When it receives the message $\mathtt{l}(a,k)$, it begins a new session n by producing the token $\mathtt{Begin}(n)$ and passing the argument a. Finally it sends the result r over k, after consuming the token $\mathtt{End}(n)$. Second, each session corresponds to a thread executing the script s. To get the correct behavior, it suffices to slightly modify the translation: tokens occurring in the translation

$$\mathcal{D}(s)_{\mathtt{Begin}(n),\mathtt{End}(n)}$$

are now parameterized with the session identifier n. The internal messages $\mathtt{Arg}(n,a)$ and $\mathtt{Return}(n,r)$ are respectively used to pass the argument a and to return the result r inside the script.

4.2 A Logic Language: Datalog with negation

Pure Criojo shares many features with logic programming. A rule with the guard `True` can be considered as an inference rule. However the premises are consumed,

as in Linear Logic: in Criojo, logical atoms are ephemeral and not persistent. It is not really a problem: just preserve the premises, by adding them to the conclusions. For instance, given a binary relation R, assume that we want to compute its reflexive and transitive closure. Here is a program in Datalog.

$$R^*(x,x) \leftarrow \texttt{True}$$
$$R^*(x,z) \leftarrow R(x,y) \wedge R^*(y,z)$$

Following the preservation principle, a first attempt to translate the second inference rule would give the following rule.

$$R(x,y), R^*(y,z) \to R(x,y), R^*(y,z), R^*(x,z)$$

However, this rule loops: an infinite number of atoms $R^*(x,z)$ can be generated. To avoid this indefinite generation, we can require that an atom is either absent in the solution, or present with a unique occurrence. Introspection can force this condition.

$$R(x,y), R^*(y,z) \to \neg\langle\, R^*(x,z)\,\rangle\, ?\, R(x,y), R^*(y,z), R^*(x,z)$$

There is still a problem. Assume we now want to compute the Cartesian product R^2 of a unary relation R, which is performed as follows in Datalog.

$$R^2(x,y) \leftarrow R(x) \wedge R(y)$$

A naive translation would give the following rule in Criojo.

$$R(x), R(y) \to \neg\langle\, R^2(x,y)\,\rangle\, ?\, R(x), R(y), R^2(x,y)$$

But this rule cannot generate $R^2(x,x)$, which requires two atoms $R(x)$ in the solution. To solve the problem, we can either increase the number of occurrences of each atom in the solution, or require a linearity condition for Datalog rules. Both options are akin. We opt for the second alternative: it forbids a rule where there are two atoms with the same predicate in the premises. The previous program in Datalog needs to be rewritten as follows.

$$R_1(x) \leftarrow R(x)$$
$$R^2(x,y) \leftarrow R(x) \wedge R_1(y)$$

The translation now works. In the following, we formalize the translation and generalize it to an extension of Datalog with negation. The implementation, which requires computing an alternating fixed point, highlights the contribution provided by the extension of Criojo with guarded commands.

The alternating fixed point construction A program in Datalog with negation is a set of inference rules. An inference rule is of the form $a \leftarrow l_1 \wedge \ldots \wedge l_n$, where the head a is an atom and where each literal l_i in the body is either a positive literal, that is an atom a_i, or a negative literal, that is the negation $\neg a_i$ of an atom a_i. It is a logical implication, asserting that from premises $l_1, \ldots, l_n$, you can deduce conclusion a. A fact is represented by a rule $a \leftarrow \texttt{True}$, called an axiom. Atoms are defined from predicates applied to variables and constants. As

usual, we assume that the rules are range-restricted: in any rule, each variable also occurs in the body of a positive literal. The condition ensures the existence of a finite set, such that each variable takes its value in this set. This finite set is a subset of the *universe*, which is the finite set of all the constants occurring in the program. Moreover, without loss of generality, we assume that two positive literals in the body never have the same predicate, likewise for two negative literals. In the following, we refer to this assumption as the linearity hypothesis.

Whereas Datalog has a univocal fixed point semantics, there are different fixed point semantics for Datalog with negation. Here, we will assign to each program its well-founded model [19], which can be characterized by an alternating fixed point construction [18]. In this model, a ground atom is either true, false or unknown. In order to compute the set of true atoms, two approximations are computed. The first one computes the set of atoms that are certainly true: this is an under-approximation. The second one computes the set of atoms that are possibly true: this is an over-approximation. By complementation, we get the set of atoms that are possibly false and certainly false respectively: an atom is possibly false if and only if it is not certainly true, and certainly false if and only if it is not possibly true. More formally, given a program D in Datalog with negation, let $\mathcal{U}$ be the finite universe, and $\mathcal{H}$ be the Herbrand base, the finite set of all the ground atoms defined from the predicates in D and the constants in $\mathcal{U}$. We define the immediate consequence operator as follows, for any set of negative ground literals N and positive ground literals A:

$$\Phi_D[N](A) \stackrel{\texttt{def}}{=} \{a[\tau] \mid \exists\, r \in D\,.\, r = a \leftarrow l_1 \wedge \ldots \wedge l_n, \forall\, i \in \{1,\ldots,n\}\,.\, l_i[\tau] \in N + A\}$$

We introduce four sequences $(\mathtt{C}_i^+)_i, (\mathtt{C}_i^-)_i, (\mathtt{P}_i^+)_i, (\mathtt{P}_i^-)_i$, with the following interpretation:

- $\mathtt{C}_i^+$: set of ground atoms that are certainly true at rank i
- $\mathtt{C}_i^-$: set of ground atoms that are certainly false at rank i
- $\mathtt{P}_i^+$: set of ground atoms that are possibly true at rank i
- $\mathtt{P}_i^-$: set of ground atoms that are possibly false at rank i

These sequences are inductively defined as follows. We use some usual notations: $\overline{X}$ denotes the complement of the set X in $\mathcal{H}$, and $\neg X$ the set of the negations of atoms in X; $\mathrm{lfp}(\varphi)$ denotes the least fixed point of the operator φ defined over the powerset $2^{\mathcal{H}}$.

$$\begin{array}{ll} \mathtt{C}_0^+ = \mathtt{C}_0^- = \emptyset & \mathtt{P}_0^+ = \mathtt{P}_0^- = \mathcal{H} \\ \mathtt{C}_{i+1}^+ = \mathrm{lfp}(\Phi_D[\neg \mathtt{C}_i^-]) & \mathtt{P}_{i+1}^+ = \mathrm{lfp}(\Phi_D[\neg \mathtt{P}_i^-]) \\ \mathtt{C}_{i+1}^- = \overline{\mathtt{P}_{i+1}^+} & \mathtt{P}_{i+1}^- = \overline{\mathtt{C}_{i+1}^+} \end{array}$$

The computation halts when the four sequences become stationary. They are ultimately stationary since the sequences $(\mathtt{C}_i^+)_i$ and $(\mathtt{C}_i^-)_i$ are increasing whereas the sequences $(\mathtt{P}_i^+)_i$ and $(\mathtt{P}_i^-)_i$ are decreasing. The well-founded model is derived from the limits $\mathtt{C}^+$ and $\mathtt{C}^-$ of the sequences $(\mathtt{C}_i^+)_i$ and $(\mathtt{C}_i^-)_i$:

- $\mathtt{C}^+$ is the set of ground atoms that are true in the model,
- $\mathtt{C}^-$ is the set of ground atoms that are false in the model.

Fig. 3. The Alternating Fixed Point Construction

Implementation in Pure Criojo To implement the alternating fixed point construction in pure Criojo, we prefer to modify the inductive definition by adding difference sequences $(\Delta\mathtt{C}_i^+)_i, (\Delta\mathtt{C}_i^-)_i, (\Delta\mathtt{P}_i^+)_i, (\Delta\mathtt{P}_i^-)_i$. Indeed, they simplify the computations by allowing the sequences $(\mathtt{C}_i^-)_i$ and $(\mathtt{P}_i^-)_i$ to be incrementally computed. See Figure 3 for a visual representation of the following equations.

$$\begin{array}{ccc} \mathtt{C}_0^+ = \mathtt{C}_0^- = \emptyset & & \mathtt{P}_0^+ = \mathtt{P}_0^- = \mathcal{H} \\ \mathtt{C}_{i+1}^+ = \mathrm{lfp}(\Phi_D[\neg\mathtt{C}_i^-]) & \Delta\mathtt{C}_{i+1}^+ = \mathtt{C}_{i+1}^+ - \mathtt{C}_i^+ & \Delta\mathtt{P}_{i+1}^- = \Delta\mathtt{C}_{i+1}^+ \\ \mathtt{P}_{i+1}^+ = \mathrm{lfp}(\Phi_D[\neg\mathtt{P}_i^-]) & \Delta\mathtt{P}_{i+1}^+ = \mathtt{P}_i^+ - \mathtt{P}_{i+1}^+ & \Delta\mathtt{C}_{i+1}^- = \Delta\mathtt{P}_{i+1}^+ \\ \mathtt{C}_{i+1}^- = \mathtt{C}_i^- + \Delta\mathtt{C}_{i+1}^- & & \mathtt{P}_{i+1}^- = \mathtt{P}_i^- - \Delta\mathtt{P}_{i+1}^- \end{array}$$

The local state is defined by using the following relations. Let m be the maximal arity of the predicates R occurring in the program D.

- $\mathtt{U}$: relation representing the universe $\mathcal{U}$
- $\mathtt{H}_n$: relation representing the Cartesian product $\mathcal{U}^n$ $(n \leq m)$
- For each predicate R occurring in the program D, relations

  $$\mathtt{C}^+[R], \mathtt{C}^-[R], \mathtt{P}^+[R], \mathtt{P}^-[R], \Delta\mathtt{C}^+[R], \Delta\mathtt{C}^-[R], \Delta\mathtt{P}^+[R], \Delta\mathtt{P}^-[R]$$

  corresponding to the projection of the sets

  $$\mathtt{C}_i^+, \mathtt{C}_i^-, \mathtt{P}_i^+, \mathtt{P}_i^-, \Delta\mathtt{C}_i^+, \Delta\mathtt{C}_i^-, \Delta\mathtt{P}_i^+, \Delta\mathtt{P}_i^-$$

  over the predicate R
- For each predicate R occurring in the program D, relations $\mathtt{C}[R]$ and $\mathtt{P}[R]$ corresponding to the result of the computation of the least fixed points at each iteration

We now describe the script computing the alternating fixed point in a sequence of elementary steps, which will be finally combined.

1. Universe Initialization

The solution is initialized with the multiset

$$\langle\, \mathtt{U}(v) \mid v \in \mathcal{U} \,\rangle\,.$$

2. Multiple Universe
Actually, we need multiple occurrences of atoms $\mathtt{U}(v)$ in order to compute the Herbrand base, used to initialize $\mathtt{P}_0^+$ and $\mathtt{P}_0^-$. The number of occurrences of each atom becomes m, the maximal arity.

$$[\mathtt{Multiplicity}] \quad \mathtt{U}(x) \multimap \neg \langle\, \mathtt{U}(x)^m \,\rangle \,?\, \mathtt{U}(x)^m$$

3. Generation of Cartesian Products (for $n \leq m$)
We also need to compute Cartesian products in order to compute the Herbrand base.

$$[\mathtt{Base}_n] \quad \mathtt{U}(x_1), \ldots, \mathtt{U}(x_n) \multimap \neg \langle\, \mathtt{H}_n(x_1, \ldots, x_n) \,\rangle \,?\, \mathtt{H}_n(x_1, \ldots, x_n), \mathtt{U}(x_1), \ldots, \mathtt{U}(x_n)$$

4. Initialization of Relations for Possible Facts (n: arity of R)

$$[\mathtt{Init}_R] \quad \mathtt{H}_n(\overrightarrow{x}) \multimap \neg \langle\, \mathtt{P}^+[R](\overrightarrow{x}) \,\rangle \,?\, \mathtt{P}^+[R](\overrightarrow{x}), \mathtt{P}^-[R](\overrightarrow{x}), \mathtt{H}_n(\overrightarrow{x})$$

This set of rules initializes the families $(\mathtt{P}^+[R])$ and $(\mathtt{P}^-[R])$ with $\mathtt{P}_0^+$ and $\mathtt{P}_0^-$ respectively.

5. Cleanup
The atoms that are henceforth useless are consumed.

$$[\mathtt{Cleanup}_n] \qquad \begin{array}{r} \mathtt{U}(x)^m \multimap \mathtt{U}(x) \\ \parallel \mathtt{H}_n(\overrightarrow{x}) \multimap \emptyset \end{array}$$

6. Computation of the Least Fixed Points
The program D in Datalog is translated into a program in Criojo: each rule in Datalog gives rise to two rules, for certain facts and possible facts respectively.

$$\mathcal{T}(D) = \|_{r \in D}\, \mathcal{T}(r)$$
$$\mathcal{T}(r) = \mathcal{T}^{\mathtt{C}}_{\mathrm{Prem}}(r) \multimap \mathcal{T}^{\mathtt{C}}_{\mathrm{Guard}}(r) \,?\, \mathcal{T}^{\mathtt{C}}_{\mathrm{Ccl}}(r) \parallel \mathcal{T}^{\mathtt{P}}_{\mathrm{Prem}}(r) \multimap \mathcal{T}^{\mathtt{P}}_{\mathrm{Guard}}(r) \,?\, \mathcal{T}^{\mathtt{P}}_{\mathrm{Ccl}}(r)$$

We now detail how to get the premises, the guard and the conclusions of the rules in Criojo. The translation is parameterized by X, either $\mathtt{C}$ for certain facts, or $\mathtt{P}$ for possible facts. The positive atoms occurring as premises are translated using the family $(X[R])$ whereas the negative atoms are translated using the family $(X^-[R])$.

$$\begin{array}{ll} \mathcal{T}^X_{\mathrm{Prem}}(a \leftarrow \overrightarrow{l}) = \mathcal{T}^X_{\mathrm{Prem}}(\overrightarrow{l}) & \\ \mathcal{T}^X_{\mathrm{Prem}}(R(\overrightarrow{t})) = X[R](\overrightarrow{t}) & \mathcal{T}^X_{\mathrm{Prem}}(\neg R(\overrightarrow{t})) = X^-[R](\overrightarrow{t}) \\ \mathcal{T}^X_{\mathrm{Prem}}(l_1 \wedge \overrightarrow{l_2}) = \mathcal{T}^X_{\mathrm{Prem}}(l_1), \mathcal{T}^X_{\mathrm{Prem}}(\overrightarrow{l_2}) & \mathcal{T}^X_{\mathrm{Prem}}(\mathtt{True}) = \emptyset \end{array}$$

If the rule r has $R(\overrightarrow{t})$ as head, then the guard tests the absence of $X[R](\overrightarrow{t})$ before the conclusion not only generates $X[R](\overrightarrow{t})$ corresponding to the head atom but also preserves all the premises.

$$\mathcal{T}^X_{\mathrm{Guard}}(r) = \neg \langle\, X[R](\overrightarrow{t}) \,\rangle \qquad \mathcal{T}^X_{\mathrm{Ccl}}(r) = \mathcal{T}^X_{\mathrm{Prem}}(r), X[R](\overrightarrow{t})$$

Finally, starting from the families $\mathtt{C}^-[R]$ and $\mathtt{P}^-[R]$ initialized with $\mathtt{C}_i^-$ and $\mathtt{P}_i^-$ respectively, the previous rules (after iterations) compute the families $\mathtt{C}[R]$ and $\mathtt{P}[R]$, equal to $\mathrm{lfp}(\Phi_D[\neg \mathtt{C}_i^-])$ and $\mathrm{lfp}(\Phi_D[\neg \mathtt{P}_i^-])$ respectively.

7. Update of Families $\mathtt{C}_i^+, \mathtt{P}_i^+, \Delta\mathtt{C}_i^+, \Delta\mathtt{P}_i^+$

The differences for positive atoms can be computed as follows.

$$\begin{array}{rl} [\mathtt{UpdateOld}_R^+] & \mathtt{C}^+[R](\overrightarrow{x}), \mathtt{C}[R](\overrightarrow{x}) \multimap \mathtt{C}^+[R](\overrightarrow{x}) \\ & \| \quad \mathtt{P}^+[R](\overrightarrow{x}) \multimap \Delta\mathtt{P}^+[R](\overrightarrow{x}) \end{array}$$

$$\begin{array}{rl} [\mathtt{UpdateDiff}_R^+] & \mathtt{C}[R](\overrightarrow{x}) \multimap \Delta\mathtt{C}^+[R](\overrightarrow{x}), \mathtt{C}^+[R](\overrightarrow{x}) \\ & \| \ \Delta\mathtt{P}^+[R](\overrightarrow{x}), \mathtt{P}[R](\overrightarrow{x}) \multimap \mathtt{P}^+[R](\overrightarrow{x}) \end{array}$$

Assume that before the execution of the previous rules, we have the following equalities.

Family	Sequence
$(\mathtt{C}[R])$	$\mathrm{lfp}(\Phi_D[\neg\mathtt{C}_i^-])$
$(\mathtt{P}[R])$	$\mathrm{lfp}(\Phi_D[\neg\mathtt{P}_i^-])$
$(\mathtt{C}^+[R])$	$\mathtt{C}_i^+$
$(\Delta\mathtt{C}^+[R])$	$\emptyset$
$(\mathtt{P}^+[R])$	$\mathtt{P}_i^+$
$(\Delta\mathtt{P}^+[R])$	$\emptyset$

Then, after their execution, we have the following equalities.

Family	Sequence
$(\mathtt{C}[R])$	$\emptyset$
$\mathtt{P}[R]$	$\emptyset$
$(\mathtt{C}^+[R])$	$\mathtt{C}_{i+1}^+$
$(\Delta\mathtt{C}^+[R])$	$\Delta\mathtt{C}_{i+1}^+$
$(\mathtt{P}^+[R])$	$\mathtt{P}_{i+1}^+$
$(\Delta\mathtt{P}^+[R])$	$\Delta\mathtt{P}_{i+1}^+$

8. Progress Condition

The computation has progressed if the following condition is satisfied.

$$\bigvee_R \exists\, \overrightarrow{x} \,.\, \Delta\mathtt{C}^+[R](\overrightarrow{x}) \vee \Delta\mathtt{P}^+[R](\overrightarrow{x})$$

Indeed, it means that some new fact has been generated by the last computation of the fixed points.

9. Update of Families $\Delta\mathtt{C}_i^-, \Delta\mathtt{P}_i^-, \mathtt{C}_i^-, \mathtt{P}_i^-,$

The differences for negative atoms can be computed as follows.

$$\begin{array}{rl} [\mathtt{UpdateDiff}_R^-] & \Delta\mathtt{C}^+[R](\overrightarrow{x}) \multimap \Delta\mathtt{P}^-[R](\overrightarrow{x}) \\ & \| \ \Delta\mathtt{P}^+[R](\overrightarrow{x}) \multimap \Delta\mathtt{C}^-[R](\overrightarrow{x}) \end{array}$$

$$\begin{array}{rl} [\mathtt{UpdateNew}_R^-] & \Delta\mathtt{C}^-[R](\overrightarrow{x}) \multimap \mathtt{C}^-[R](\overrightarrow{x}) \\ & \| \ \Delta\mathtt{P}^-[R](\overrightarrow{x}), \mathtt{P}^-[R](\overrightarrow{x}) \multimap \end{array}$$

Assume that before the execution of the previous rules, we have the following equalities.

Family	Sequence
$(\Delta\mathtt{C}^+[R])$	$\Delta\mathtt{C}_{i+1}^+$
$(\Delta\mathtt{P}^-[R])$	$\emptyset$
$(\Delta\mathtt{P}^+[R])$	$\Delta\mathtt{P}_{i+1}^+$
$(\Delta\mathtt{C}^-[R])$	$\emptyset$
$(\mathtt{P}^-[R])$	$\mathtt{P}_i^-$
$(\mathtt{C}^-[R])$	$\mathtt{C}_i^-$

Then, after their execution, we have the following equalities.

Family	Sequence
$(\Delta\mathtt{C}^+[R])$	$\emptyset$
$(\Delta\mathtt{P}^-[R])$	$\emptyset$
$(\Delta\mathtt{P}^+[R])$	$\emptyset$
$(\Delta\mathtt{C}^-[R])$	$\emptyset$
$(\mathtt{P}^-[R])$	$\mathtt{P}^-_{i+1}$
$(\mathtt{C}^-[R])$	$\mathtt{C}^-_{i+1}$

Thus a new computation for the fixed points is possible.

Finally, we can conclude.

Theorem 4 (Alternating Fixed Point in Pure Criojo). *Let D be a Datalog program with universe $\mathcal{U}$ and Herbrand Base $\mathcal{H}$. Let us define a script D' in the Criojo variant of Dijkstra's language of guarded commands as follows.*

$\mathtt{do}\,\{\mathtt{Multiplicity}\};$ *(Multiple Universe)*
$\mathtt{do}\,\{\|_n\,\mathtt{Base}_n\};$ *(Cartesian Products for Herbrand base)*
$\mathtt{do}\,\{\|_R\,\mathtt{Init}_R\};$ *(Initialization)*
$\mathtt{do}\,\{\|_n\,\mathtt{Cleanup}_n\};$ *(Cleanup)*
$\mathtt{do}\,\{\mathcal{T}(D)\};$ *(First Fixed-Point Computation)*
$\mathtt{do}\,\{\|_R\,\mathtt{UpdateOld}^+_R\};\mathtt{do}\,\{\|_R\,\mathtt{UpdateDiff}^+_R\};$ *(First Update of Positive Families)*
$\mathtt{do}\,\{$ *(Main Loop)*
$\quad\big(\bigvee_R \exists\,\overrightarrow{x}\,.\,\Delta\mathtt{C}^+[R](\overrightarrow{x}) \vee \Delta\mathtt{P}^+[R](\overrightarrow{x})\big)\triangleright$ *(If Progress)*
$\quad\mathtt{do}\,\{\|_R\,\mathtt{UpdateDiff}^-_R\};\mathtt{do}\,\{\|_R\,\mathtt{UpdateNew}^-_R\};$ *(Update of Negative Families)*
$\quad\mathtt{do}\,\{\mathcal{T}(D)\};$ *(New Fixed-Point Computation)*
$\quad\mathtt{do}\,\{\|_R\,\mathtt{UpdateOld}^+_R\};\mathtt{do}\,\{\|_R\,\mathtt{UpdateDiff}^+_R\};$ *(New Update of Positive Families)*
$\}$ *(End of Loop)*

Then the script D' transforms the solution

$$\langle\,\mathtt{U}(v)\mid v\in\mathcal{U}\,\rangle$$

into the solution

$$\langle\,\mathtt{U}(v)\mid v\in\mathcal{U}\,\rangle\uplus(\biguplus_R\mathtt{C}^+[R])\uplus(\biguplus_R\mathtt{C}^-[R])\uplus(\biguplus_R\mathtt{P}^+[R])\uplus(\biguplus_R\mathtt{P}^-[R])$$

where the families $(\mathtt{C}^+[R])$ and $(\mathtt{C}^-[R])$ are equal to $\mathtt{C}^+$ and $\mathtt{C}^-$, the set of ground atoms that are true and false respectively in the well-founded model, and where the families $(\mathtt{P}^-[R])$ and $(\mathtt{P}^+[R])$ are equal to their complementary with respect to the Herbrand base $\mathcal{H}$.

Proof. By induction on the index i used for iterations. The result derives from the description previously given of the different steps involved in the script. □

4.3 A Functional Language: Gödel's System T

As an idiomatic language for functional programming, we consider Gödel's System T [20, chap. 7], a λ-calculus with inductive types and recursion operators. Here we only consider the type of natural numbers, directly represented by constructors 0 and $\mathtt{s}$, and use a call-by-value and weak (without reduction under

abstraction) strategy. We do not consider the type system, which ensures that the redexes effectively reduce while preserving typing. In order to cope with distribution, we extend the language by adding the possibility to call external functions.

$$
\begin{array}{lll}
\text{Term} & e & ::= x \mid \lambda x\,.\,e \mid e\,e \mid 0 \mid \mathtt{s}(e) \mid \mathtt{r}(e,e,e) \mid f \\
\text{Value} & v & ::= \lambda x\,.\,e \mid 0 \mid \mathtt{s}(v) \mid f \\
\text{Redex} & r & ::= v\,v \mid \mathtt{r}(v,v,v) \\
\text{Reduction Context} & E & ::= \ - \ \mid E\,e \mid v\,E \mid \mathtt{s}(E) \mid \mathtt{r}(E,e,e) \mid \mathtt{r}(v,E,e) \mid \mathtt{r}(v,v,E)
\end{array}
$$

Since we need to represent the language in Criojo, its (nominal) signature is added to the signature used in Criojo. However, we keep the usual notation for λ-terms, instead of the notation for nominal terms.

Thanks to the impure version of Criojo, we can embed Gödel's System T into Criojo in a straightforward way, leading to a distributed version for free. Indeed, the operational semantics, expressed by an inference system, can be considered as a finite description of local rules for the cham. The state of the agent $\mathtt{ST_i}$, implementing some function $\mathtt{c}$, is represented as an aggregate combining (i) a global counter $\mathtt{Session}(n)$ for the identifiers of the internal sessions, and (ii) pending requests. A pending request $\mathtt{LC}(n,e)\,\&\,\mathtt{Rep}(n,k,s)$ indicates that in the internal session n, the agent is currently evaluating e and will response over channel k with external identifier s. The join operator & is assumed as usual to be associative and commutative, which leads to the following rules.

$$
\sigma_1\,\&(\sigma_2\,\&\,\sigma_3) \equiv (\sigma_1\,\&\,\sigma_2)\,\&\,\sigma_3 \qquad \sigma_1\,\&\,\sigma_2 \equiv \sigma_2\,\&\,\sigma_1
$$

$$
\frac{\sigma_1 \equiv \sigma_1' \quad \mathtt{ST_i} \vdash \sigma_1' \to \sigma_2' \quad \sigma_2' \equiv \sigma_2}{\mathtt{ST_i} \vdash \sigma_1 \to \sigma_2}
$$

The agent $\mathtt{ST_i}$ provides a channel $\mathtt{g}$ for requests. A client transmits the argument a of the function $\mathtt{c}$, a channel k to get the response and an external identifier s used to correlate the response with the calling computation. At each request, a fresh session identifier is used, allowing multiple computations to be run in parallel. The response is finally sent over the external channel, with the external identifier.

$$
\mathtt{ST_i} \vdash \sigma\,\&\,\mathtt{Session}(n), \mathtt{g}(a,k,s) \to \mathtt{LC}(n,\mathtt{c}\,a)\,\&\,\mathtt{Rep}(n,k,s)\,\&\,\sigma\,\&\,\mathtt{Session}(\mathtt{succ}(n))
$$

$$
\big(\mathtt{ST_i} \vdash \mathtt{LC}(n,v)\,\&\,\mathtt{Rep}(n,k,s)\,\&\,\sigma \to \sigma, k(s,v)\big)_{\,v \text{ value}}
$$

Note that the second rule is indexed by an infinite set: indeed, it is impossible to match a value, because of the case $\mathtt{s}(v)$. The remaining rules directly come from the operational semantics. Again they describe an infinite set of rules.

$$
\begin{array}{l}
\big(\mathtt{ST_i} \vdash \quad \mathtt{LC}(n,(\lambda x\,.\,e)\,v)\,\&\,\sigma \to \mathtt{LC}(n,e[v/x])\,\&\,\sigma\big)_{v} \\
\big(\mathtt{ST_i} \vdash \quad \mathtt{LC}(n,\mathtt{r}(v_1,v_2,0))\,\&\,\sigma \to \mathtt{LC}(n,v_1)\,\&\,\sigma\big)_{v_1,v_2} \\
\big(\mathtt{ST_i} \vdash \mathtt{LC}(n,\mathtt{r}(v_1,v_2,\mathtt{s}(v)))\,\&\,\sigma \to \mathtt{LC}(n,v_2\,\mathtt{r}(v_1,v_2,v)\,v)\,\&\,\sigma\big)_{v_1,v_2,v} \\
\left(\dfrac{\mathtt{ST_i} \vdash \mathtt{LC}(n,e_1)\,\&\,\sigma \to \mathtt{LC}(n,e_2)\,\&\,\sigma}{\mathtt{ST_i} \vdash \mathtt{LC}(n,E[e_1])\,\&\,\sigma \to \mathtt{LC}(n,E[e_2])\,\&\,\sigma}\right)_{E}
\end{array}
$$

Finally, the rules for calling an external function follow the protocol that is now common. The agent $\mathtt{ST_i}$ sends a request, with the argument, the return channel

`retF` and the session identifier, then waits for the response on channel `retF`.

$$\begin{aligned}&\big(\mathtt{ST_i} \vdash \quad \mathtt{LC}(n, E[f\,v]) \,\&\, \sigma \to \mathtt{LC}(n, E) \,\&\, \sigma, f(v, \mathtt{retF}, n)\big)_{E,v}\\&\big(\mathtt{ST_i} \vdash \mathtt{LC}(n, E) \,\&\, \sigma, \mathtt{retF}(n, v) \to \mathtt{LC}(n, E[v]) \,\&\, \sigma\big)_{E}\end{aligned}$$

To translate the program in pure Criojo, we need to solve two problems: decomposing in a top-down way a term into a value or a redex in a reduction context, and realizing the substitution involved in the β-reduction. The local state is defined with three predicates, `LC` for terms, `V` for values, `RC` for reduction contexts. First the request-response protocol is implemented as usual.

$$\begin{aligned}&\mathtt{ST_p} \vdash \quad \mathtt{Session}(n), \mathtt{g}(a, k, s) \to \mathtt{LC}(n, \mathtt{c}\,a), \mathtt{RC}(n, \varepsilon), \mathtt{Rep}(n, k, s), \mathtt{Session}(\mathtt{succ}(n))\\&\mathtt{ST_p} \vdash \mathtt{V}(n, v), \mathtt{RC}(n, \varepsilon), \mathtt{Rep}(n, k, s) \to k(s, v)\end{aligned}$$

The reduction context $-$ is represented as ε, an empty stack. Indeed, reduction contexts are represented as stacks.

$$\text{Stack} \quad S ::= \varepsilon \mid S :: - \, e \mid S :: v \, - \mid S :: \mathtt{s}(-) \mid S :: \mathtt{r}(-, e, e) \mid S :: \mathtt{r}(v, -, e) \mid S :: \mathtt{r}(v, v, -)$$

We denote by $\overline{S}$ the reduction context associated to S. The usual unique decomposition of a term into a value or a redex in a reduction context can be rephrased as follows: for any term e, there exists a unique value v and a unique stack S of the form ε, $(S' :: v' \, -)$ or $\big(S' :: \mathtt{r}(v', v'', -)\big)$ such that $e = \overline{S}[v]$. In that case we say that (S, v) is the canonical decomposition of e. To get the decomposition, we implement a focus function, a well-known technique for abstract machines dedicated to functional languages, formalized by Danvy et al. [9]. The focus function leads to the intended canonical decomposition, in two alternating steps. The stack is progressively built, in a top-down left-to-right movement.

$$\begin{aligned}&\mathtt{ST_p} \vdash \quad \mathtt{LC}(n, e_1\,e_2), \mathtt{RC}(n, S) \to \mathtt{LC}(n, e_1), \mathtt{RC}(n, S :: - \, e_2)\\&\mathtt{ST_p} \vdash \quad \mathtt{V}(n, v), \mathtt{RC}(n, S :: - \, e) \to \mathtt{LC}(n, e), \mathtt{RC}(n, S :: v \, -)\\&\mathtt{ST_p} \vdash \quad \mathtt{LC}(n, \mathtt{s}(e)), \mathtt{RC}(n, S) \to \mathtt{LC}(n, e), \mathtt{RC}(n, S :: \mathtt{s}(-))\\&\mathtt{ST_p} \vdash \quad \mathtt{LC}(n, \mathtt{r}(e_1, e_2, e_3)), \mathtt{RC}(n, S) \to \mathtt{LC}(n, e_1), \mathtt{RC}(n, S :: \mathtt{r}(-, e_2, e_3))\\&\mathtt{ST_p} \vdash \mathtt{V}(n, v_1), \mathtt{RC}(n, S :: \mathtt{r}(-, e_2, e_3)) \to \mathtt{LC}(n, e_2), \mathtt{RC}(n, S :: \mathtt{r}(v_1, -, e_3))\\&\mathtt{ST_p} \vdash \mathtt{V}(n, v_2), \mathtt{RC}(n, S :: \mathtt{r}(v_1, -, e_3)) \to \mathtt{LC}(n, e_3), \mathtt{RC}(n, S :: \mathtt{r}(v_1, v_2, -))\end{aligned}$$

As for the values, they are built in a bottom-up movement.

$$\begin{aligned}&\mathtt{ST_p} \vdash \quad \mathtt{LC}(n, \lambda\, x \,.\, e) \to \mathtt{V}(n, \lambda\, x \,.\, e))\\&\mathtt{ST_p} \vdash \quad \mathtt{LC}(n, 0) \to \mathtt{V}(n, 0)\\&\mathtt{ST_p} \vdash \mathtt{V}(n, v), \mathtt{RC}(n, S :: \mathtt{s}(-)) \to \mathtt{V}(n, \mathtt{s}(v)), \mathtt{RC}(n, S)\\&\mathtt{ST_p} \vdash \quad \mathtt{LC}(n, f) \to \mathtt{V}(n, f)\end{aligned}$$

Finally, the redexes can be reduced. For the β-reduction, an external call is needed to perform the substitution: the agent uses the channel `retS` to get the

response.

$$
\begin{aligned}
\mathtt{ST}_\mathrm{p} \vdash \quad & \mathtt{V}(n,v), \mathtt{RC}(n, S :: (\lambda\, x\, .\, e)\, -) \to \mathtt{RC}(n,S), \mathtt{doSub}(e,v,x,\mathtt{retS},n) \\
\mathtt{ST}_\mathrm{p} \vdash \quad & \mathtt{retS}(n,e) \to \mathtt{LC}(n,e) \\
\mathtt{ST}_\mathrm{p} \vdash \quad & \mathtt{V}(n,0), \mathtt{RC}(n, S :: \mathtt{r}(v_1,v_2,-)) \to \mathtt{V}(n,v_1), \mathtt{RC}(n,S) \\
\mathtt{ST}_\mathrm{p} \vdash \quad & \mathtt{V}(n,\mathtt{s}(v)), \mathtt{RC}(n, S :: \mathtt{r}(v_1,v_2,-)) \to \mathtt{LC}(n, v_2\, \mathtt{r}(v_1,v_2,v)\, v), \mathtt{RC}(n,S) \\
\mathtt{ST}_\mathrm{p} \vdash \quad & \mathtt{V}(n,v), \mathtt{RC}(n, S :: f\, -) \to \mathtt{RC}(n,S), f(v,\mathtt{retF},n) \\
\mathtt{ST}_\mathrm{p} \vdash \quad & \mathtt{retF}(n,v) \to \mathtt{V}(n,v)
\end{aligned}
$$

Both implementations, in impure Criojo and in pure Criojo respectively, are equivalent.

Theorem 5 (Gödel's System T – Bisimilarity). *Let K be the following set of channels:*

$$\mathtt{doSub}, \mathtt{retS}, \mathtt{isEqual}, \mathtt{equal}^+, \mathtt{equal}^-.$$

The collaborations

$$\langle\, \mathtt{ST}_\mathrm{i}[\langle\, \mathtt{Session}(0)\,\rangle]\,\rangle \setminus \emptyset$$

and

$$\langle\, \mathtt{ST}_\mathrm{p}[\langle\, \mathtt{Session}(0)\,\rangle], \mathtt{SUB}_\mathrm{p}[\langle\, \mathtt{Session}(0), \mathtt{RecCall}(0)\,\rangle], \mathtt{EQ}[\langle\, \mathtt{Session}(0)\,\rangle]\,\rangle \setminus K$$

are bisimilar.

Proof. We sketch the proof.

We split into four sets the relations and channels of $\mathtt{ST}_\mathrm{p}$ and $\mathtt{ST}_\mathrm{i}$:

$$
\begin{aligned}
G &= \{\mathtt{Session}, \mathtt{Rep}\}, \\
H &= \{\mathtt{LC}, \mathtt{V}, \mathtt{RC}, \mathtt{doSub}, \mathtt{retS}\}, \\
I &= \{\mathtt{g}, k, f, \mathtt{retF}\}.
\end{aligned}
$$

We also consider a restriction $\mathtt{ST}'_\mathrm{p}$ of agent $\mathtt{ST}_\mathrm{p}$: it contains all the rules of the focus function, as well as the rule $\mathtt{ST}'_\mathrm{p} \vdash \mathtt{retS}(n,e) \to \mathtt{LC}(n,e)$. In other words, the agent $\mathtt{ST}'_\mathrm{p}$ performs the decomposition needed before reduction.

We exhibit a bisimulation R containing both collaborations. Consider collaborations

$$\langle\, \mathtt{ST}_\mathrm{i}[\langle\, A_1 \,\&\, B_1\,\rangle \uplus C_1]\,\rangle \setminus \emptyset$$

and

$$(\langle\, \mathtt{ST}_\mathrm{p}[A_2 \uplus B_2 \uplus C_2], \mathtt{SUB}_\mathrm{p}[E_2], \mathtt{EQ}[F_2]\,\rangle \uplus M_2) \setminus K$$

where

- A_1 and A_2 contain the messages over G,
- B_1 and B_2 contain the messages over H,
- C_1 and C_2 contain the messages over I,
- M_2 contains the messages in transit, therefore over K.

They are related by R if the following conditions are satisfied.

(A) There is a biunivocal correspondence between the impure state A_1 and the pure state A_2, and between the external messages C_1 and C_2, built as follows.

$$\begin{aligned}
\mathtt{Rep}(n,k,s) &\longleftrightarrow \mathtt{Rep}(n,k,s)\\
\mathtt{Session}(n) &\longleftrightarrow \mathtt{Session}(n)\\
\mathtt{g}(a,k,s) &\longleftrightarrow \mathtt{g}(a,k,s)\\
k(s,v) &\longleftrightarrow k(s,v)\\
f(v,\mathtt{retF},n) &\longleftrightarrow f(v,\mathtt{retF},n)\\
\mathtt{retF}(n,v) &\longleftrightarrow \mathtt{retF}(n,v)
\end{aligned}$$

(B) There exists two sets X and Y of session identifiers, a bipartition (B_1', B_1'') of B_1 and a bipartition (B_2', B_2'') of B_2 such that
- $B_1' = \langle\, \mathtt{LC}(n, E_n) \mid n \in X \,\rangle$,
- $B_1'' = \langle\, \mathtt{LC}(n, e_n) \mid n \in Y \,\rangle$,
- $B_2' = \langle\, \mathtt{RC}(n, S_n') \mid n \in X \,\rangle$,
- $B_2'' = B_2 - B_2'$.

Moreover, the following conditions are satisfied:
- there is a biunivocal correspondence between B_1' and B_2', with for all n in X, $\overline{S_n'} = E_n$,
- the collaboration

$$\langle\, \mathtt{ST}_{\mathrm{p}}'[B_2''], \mathtt{SUB}_{\mathrm{p}}[E_2], \mathtt{EQ}[F_2] \,\rangle \uplus M_2$$

converges to the following final state $\langle\, \mathtt{V}(n, v_n), \mathtt{RC}(n, S_n'') \mid n \in Y \,\rangle$, such that for any n in Y, (S_n'', v_n) is the canonical decomposition of e_n.

Finally, it remains to prove that R is a bisimulation. We need to consider three cases, corresponding to the silent action, an input message (over channels `retF` or `g`) and an output message (over channels f or k). We deduce the simulation properties from the following lemmas.

(1) First Lemma – Canonical Decomposition
Assume the collaboration

$$\langle\, \mathtt{ST}_{\mathrm{p}}'[B], \mathtt{SUB}_{\mathrm{p}}[E], \mathtt{EQ}[F] \,\rangle \uplus M$$

converges to the following final state

$$\langle\, \mathtt{V}(n, v_n), \mathtt{RC}(n, S_n) \mid n \in Y \,\rangle,$$

such that for any n in Y, (S_n, v_n) is the canonical decomposition of some term e_n. Then given a fresh session identifier m, the collaboration

$$\langle\, \mathtt{ST}_{\mathrm{p}}'[B \uplus \langle\, \mathtt{LC}(m, e), \mathtt{RC}(m, S) \,\rangle], \mathtt{SUB}_{\mathrm{p}}[E], \mathtt{EQ}[F] \,\rangle \uplus M$$

converges to the following final state

$$\langle\, \mathtt{V}(n, v_n), \mathtt{RC}(n, S_n) \mid n \in Y \,\rangle \uplus \langle\, \mathtt{V}(m, v'), \mathtt{RC}(m, S') \,\rangle,$$

such that (S', v') is the canonical decomposition of $\overline{S}[e]$.
It is sufficient to prove that the rules defining $\mathtt{ST}_{\mathrm{p}}'$ are convergent. Indeed, the normal forms clearly correspond to the canonical decomposition. First, when we consider a unique session, the rules are deterministic. Second, there is a natural well-founded order over reductions: it measures the length of the traversal, which is depth-first and left-to-right.

(2) Second Lemma – Operational Simulation
Starting from two collaborations

$$\Lambda_1 \stackrel{\text{def}}{=} \langle\, \mathtt{ST_i}[\langle\, A_1 \,\&\, B_1 \,\rangle \uplus C_1] \,\rangle \text{ and } \Lambda_2 \stackrel{\text{def}}{=} (\langle\, \mathtt{ST_p}[A_2 \uplus B_2 \uplus C_2], \mathtt{SUB_p}[E_2], \mathtt{EQ}[F_2] \,\rangle \uplus M_2)$$

such that $(\Lambda_1 \setminus \emptyset, \Lambda_2 \setminus K)$ belongs to R, if $\Lambda_1 \to \Lambda_1'$, then there exists Λ_2' such that $\Lambda_2 \to^* \Lambda_2'$ and $(\Lambda_1' \setminus \emptyset, \Lambda_2' \setminus K)$ belongs to R, and conversely if $\Lambda_2 \to \Lambda_2'$, then there exists Λ_1' such that $\Lambda_1 \to^* \Lambda_1'$ and $(\Lambda_1' \setminus \emptyset, \Lambda_2' \setminus K)$ belongs to R. By a case analysis over reductions and by using the preceding lemma.

□

4.4 A Concurrent Language: The π-Calculus

As an idiomatic language for concurrent programming, we consider the asynchronous π-calculus [32, chap. 5], defined as follows.

$$p \;::=\; 0 \mid p \parallel p \mid \overline{x}\, y \mid x(y).p \mid !x(y).p \mid \nu x.p$$

Without loss of generality, we opt for input replication instead of general replication. Moreover, we omit sums, corresponding to external choices, to simplify the presentation since their translation is a bit lengthy. In order to avoid confusions between Criojo channels and π-calculus channels, a channel of the π-calculus is called a name in the following, whereas a channel always refer to a Criojo channel. Since we need to represent the language in Criojo, its (nominal) signature is added to the signature used in Criojo. However, we keep the usual notation for π-processes, instead of the notation for nominal terms.

As with Gödel's System T, thanks to the impure version of Criojo, we can embed the π-calculus in Criojo in a straightforward way, leading to a distributed version: just translate the operational semantics into rules for the cham. The state of an agent that implements a process of the π-calculus is represented as a pair $\mathtt{Pi}(p) \,\&\, \mathtt{New}(n)$, where p is the current process and n is a counter generating new identifiers, allowing the creation of new names. Each agent provides a unique channel. A name is represented as an ordered pair (k, n), where k is a channel provided by an agent and n is an identifier generated by the agent. Thus a π-calculus particle $\overline{(k,n)}\, x$ corresponds to the Criojo message $k(n, x)$. There are two related rules, for distribution. Although they involve the nil process, they are actually general, since any process p can be put in parallel, by applying an inference rule defined below.

$$\mathtt{PI_i} \vdash \mathtt{Pi}(0) \,\&\, \mathtt{New}(n), \mathtt{l}(m, x) \to \mathtt{Pi}(\overline{(\mathtt{l}, m)}\, x) \,\&\, \mathtt{New}(n) \qquad (\mathtt{l} \in \mathcal{K}(\mathtt{PI_i}))$$

$$\mathtt{PI_i} \vdash \quad \mathtt{Pi}(\overline{(k, m)}\, x) \,\&\, \mathtt{New}(n) \to \mathtt{Pi}(0) \,\&\, \mathtt{New}(n), k(m, x) \qquad (k \notin \mathcal{K}(\mathtt{PI_i}))$$

The following rules directly express the reduction axioms. They correspond to a finite representation of an infinite set of rules, because of substitutions. Note that we use a name server to manage scope extrusion, as proposed by Berry and Boudol [2].

$$\big(\mathtt{PI_i} \vdash \qquad \mathtt{Pi}(\nu x.p) \,\&\, \mathtt{New}(n) \to \mathtt{Pi}(p[(\mathtt{l}, n)/x]) \,\&\, \mathtt{New}(\mathtt{succ}(n))\big)_p$$

$$\big(\mathtt{PI_i} \vdash \mathtt{Pi}(\overline{x}\, y' \parallel x(y).p) \,\&\, \mathtt{New}(n) \to \mathtt{Pi}(p[y'/y]) \,\&\, \mathtt{New}(n)\big)_p$$

$$\big(\mathtt{PI_i} \vdash \mathtt{Pi}(\overline{x}\, y' \parallel !x(y).p) \,\&\, \mathtt{New}(n) \to \mathtt{Pi}(p[y'/y] \parallel !x(y).p) \,\&\, \mathtt{New}(n)\big)_p$$

There are also the standard rules for the congruence relation, expressing that the processes equipped with the nil process and the parallel operator form a commutative monoid.

$$p \parallel 0 \equiv p \qquad p \parallel p' \equiv p' \parallel p \qquad p \parallel (p' \parallel p'') \equiv (p \parallel p') \parallel p''$$

Finally, two inference rules express that the congruence relation and the parallel operator are compatible with the reduction relation.

$$\frac{q \equiv p \qquad \mathtt{PI_i} \vdash \mathtt{Pi}(p) \,\&\, \mathtt{New}(n), \overrightarrow{\mathtt{l}(v)} \to \mathtt{Pi}(p') \,\&\, \mathtt{New}(n'), \overrightarrow{k'(v')} \qquad p' \equiv q'}{\begin{array}{c} \mathtt{PI_i} \vdash \mathtt{Pi}(q) \,\&\, \mathtt{New}(n), \overrightarrow{\mathtt{l}(v)} \to \mathtt{Pi}(q') \,\&\, \mathtt{New}(n'), \overrightarrow{k'(v')} \\ \mathtt{PI_i} \vdash \mathtt{Pi}(p) \,\&\, \mathtt{New}(n), \overrightarrow{\mathtt{l}(v)} \to \mathtt{Pi}(p') \,\&\, \mathtt{New}(n'), \overrightarrow{k'(v')} \\ \hline \mathtt{PI_i} \vdash \mathtt{Pi}(p \parallel q) \,\&\, \mathtt{New}(n), \overrightarrow{\mathtt{l}(v)} \to \mathtt{Pi}(p' \parallel q) \,\&\, \mathtt{New}(n'), \overrightarrow{k'(v')} \end{array}}$$

To translate a π-calculus process in pure Criojo, we need to solve two problems: first, realizing the substitutions involved in the different reductions, second, expressing the congruence and the compatibility rules. The solution to the first problem is easy: just use an external agent to realize substitutions, as already seen for System T. For the second problem, the solution is straightforward: the free commutative monoid of processes in parallel can be taken to be the set of finite multisets with processes as elements. That is the reason why "the chemical abstract machine can be regarded as the computational model of the π-calculus" [11]. The local state is defined with two predicates, `Pi` for processes and `New` for creating new names. Consider an agent $\mathtt{PI_p}$ providing the channel `l`. The local state $\langle\, \mathtt{Pi}(p_1), \ldots, \mathtt{Pi}(p_n), \mathtt{New}(n) \,\rangle$ represents the process $p_1 \parallel \ldots \parallel p_n$ and the fact that the next new name will be $(\mathtt{l}, n)$. Here are the two rules for distribution.

$$\begin{aligned} \mathtt{PI_p} \vdash \quad & \mathtt{l}(n, x) \to \mathtt{Pi}(\overline{(\mathtt{l}, n)}\, x) && (\mathtt{l} \in \mathcal{K}(\mathtt{PI_p})) \\ \mathtt{PI_p} \vdash \mathtt{Pi}(\overline{(k, n)}\, x) & \to k(n, x) && (k \notin \mathcal{K}(\mathtt{PI_p})) \end{aligned}$$

The following rules translate the communication rules and the rule for new names. Since they require a substitution, an external message $\mathtt{doSub}(p, y, x, \mathtt{pi})$ is generated to compute the substitution $p[y/x]$. The response will use the channel `pi`.

$$\begin{aligned} \mathtt{PI_p} \vdash \quad & \mathtt{Pi}(\nu x.p), \mathtt{New}(n) \to \mathtt{doSub}(p, (\mathtt{l}, n), x, \mathtt{pi}), \mathtt{New}(\mathtt{succ}(n)) \\ \mathtt{PI_p} \vdash \; & \mathtt{Pi}(\overline{x}\, y'), \mathtt{Pi}(x(y).p) \to \mathtt{doSub}(p, y', y, \mathtt{pi}), \\ \mathtt{PI_p} \vdash \; & \mathtt{Pi}(\overline{x}\, y'), \mathtt{Pi}(!x(y).p) \to \mathtt{doSub}(p, y', y, \mathtt{pi}), \mathtt{Pi}(!x(y).p) \\ \mathtt{PI_p} \vdash \quad & \mathtt{pi}(p) \to \mathtt{Pi}(p) \end{aligned}$$

Finally, the processes in parallel are embedded into the multiset defined by the local solution, thanks to the following decomposition rules.

$$\begin{aligned} \mathtt{PI_p} \vdash \quad & \mathtt{Pi}(0) \to \\ \mathtt{PI_p} \vdash \; & \mathtt{Pi}(p \parallel q) \to \mathtt{Pi}(p), \mathtt{Pi}(q) \end{aligned}$$

Finally, the translation is a simple adaptation of the one given by Berry and Boudol [2], with a name server to generate new names. The only difference comes

from substitutions: they are not effectively described in the original translation, which therefore contains an infinite set of rules, hence is impure following our terminology, while they are performed by an adjoint agent in pure Criojo.

Again, both implementations, in impure Criojo and in pure Criojo respectively, are equivalent.

Theorem 6 (π-calculus – Bisimilarity). *Let K be the following set of channels:*

$$\{\texttt{doSub}, \texttt{pi}, \texttt{isEqual}, \texttt{equal}^+, \texttt{equal}^-\}.$$

For any process p, the collaborations

$$\langle\, \texttt{PI}_\text{i}[\langle\, \texttt{Pi}(p), \texttt{New}(0)\,\rangle]\,\rangle \setminus \emptyset$$

and

$$\langle\, \texttt{PI}_\text{P}[\langle\, \texttt{Pi}(p), \texttt{New}(0)\,\rangle], \texttt{SUB}_\text{P}[\langle\, \texttt{Session}(0), \texttt{RecCall}(0)\,\rangle], \texttt{EQ}[\langle\, \texttt{Session}(0)\,\rangle]\,\rangle \setminus K$$

are bisimilar.

Proof. We sketch the proof.

We exhibit a bisimulation R containing both collaborations. Consider collaborations

$$\langle\, \texttt{PI}_\text{i}[C_1]\,\rangle \setminus \emptyset$$

and

$$(\langle\, \texttt{PI}_\text{P}[D_1], \texttt{SUB}_\text{P}[D_2], \texttt{EQ}[D_3]\,\rangle \uplus M_1) \setminus K$$

where M_1 is a multiset of messages over channels in K. They are related by R if the following conditions are satisfied.

(A) There is a biunivocal correspondence between the impure local solution C_1 and the pure one D_1, when considering the relation `New` and the channels `l` and k.

$$\begin{aligned} \texttt{New}(n) &\longleftrightarrow \texttt{New}(n) \\ \texttt{l}(n,x) &\longleftrightarrow \texttt{l}(n,x) \\ k(n,x) &\longleftrightarrow k(n,x) \end{aligned}$$

(B) Consider the remaining solution in C_1: $\langle\, \texttt{Pi}(p)\,\rangle$. Consider now the remaining solution in D_1. It can be split into two solutions, on the one hand $\langle\, \texttt{Pi}(p_1), \ldots, \texttt{Pi}(p_n)\,\rangle$, on the other hand M_2, a multiset of external messages over the channels `doSub` and `pi`, used for substitutions. Then the collaboration

$$\langle\, \texttt{SUB}_\text{P}[D_2], \texttt{EQ}[D_3]\,\rangle \uplus M_1 \uplus M_2$$

converges to the following final state $\langle\, \texttt{pi}(p'_1), \ldots, \texttt{pi}(p'_m)\,\rangle$, and we have

$$p \equiv (p_1 \parallel \ldots \parallel p_n) \parallel (p'_1 \parallel \ldots \parallel p'_m).$$

Finally, it remains to prove that R is a bisimulation. It is straightforward, by using a canonical decomposition for any process p: $p \equiv (p_1 \parallel \ldots \parallel p_n)$, where each process p_i is either an output particle, an input process (possibly with replication), or a restriction process. □

5 Summary – Related Work – Perspectives

The pivot language Criojo is the internal language of the chemical abstract machine that we have designed for service-oriented computing: it allows the collaborations between agents and the rules specific to each agent to be defined. In its impure form, by abstracting the state of agents, it provides a universal language for interfacing (informational or computational) resources. We have seen three examples of this ability: the translation of (i) a variant of Dijkstra's language of guarded commands, (ii) Gödel's System T and (iii) the π-calculus. For Gödel's System T and the π-calculus, the translation was eased by the algebraic framework used to represent data in Criojo. In its pure form, the language Criojo provides a universal language for orchestrating services, thanks to an extension of the chemical abstract machine by an introspection mechanism. We have seen four examples of this ability: the translation of the three previous languages as well as Datalog with negation, which covers the major paradigms in programming.

Our starting point is clearly Berry and Boudol's chemical abstract machine [2]. However this is not an effective machine. First, the reversible rules are not effective, as pointed out by Garg et al. [17]. Second, the set of specific rules may be infinite, which implies an external device generating them, leading to an impure aspect. The reflexive chemical abstract machine, proposed by Fournet and Gonthier [11], brings effectiveness, thanks to intercession (the ability to produce new reaction rules), a restriction of the algebraic signatures to relational signatures and to a linearity condition greatly simplifying pattern matching. From a framework designed to express operational semantics, the chemical abstract machine has evolved to a calculus, the join-calculus, possible core of real distributed programming languages [12]. However, since the reduction rules of the reflexive chemical abstract machine are still local, effectiveness does not imply completeness. Indeed, as proved here, such a machine cannot compute all the transformations of a chemical solution that an introspective machine can compute. If the introspective chemical abstract machine is new, the idea of introspection is not new in chemical models: see the definition of contexts as catalysts or inhibitors given by Braione and Picco [5], or the extension of the language *Constraint handling Rules* (CHR) with a negation as absence [38].

As shown by CHR, a declarative language based on multiset rewriting, originally designed for writing constraint solvers and now employed as a general purpose language [13], chemical models deal not only with concurrent and distributed computing, but also logic programming. Thus the language Criojo is also inspired by logic languages like Datalog [7]. a query language for deductive databases, in other words for structures in the relational model. However, Datalog has a major limitation: it cannot express the deletion or the update of resources. Its semantics is essentially monotone: the representation of resources always increases during computations. Linear Logic, considered as a logic for resources, has turned out to be useful for addressing the problem: Pfenning and Simmons [33] resort to linear resources to generalize the logical algorithms of Ganzinger and McAllester [16] a first attempt for solving the problem. Likewise,

Betz, Raiser and Frhwirth have developed new foundations based on Linear Logic for the language CHR [3].

The development of Criojo, as a distributed and declarative language, can also be seen as part of a recent trend exemplified by the project BOOM [1]. The trend relies on two hypotheses: (i) the design of distributed systems should be data-centric, capturing the states of a system as logical structures, like the one defined by the internal messages in Criojo, (ii) the behavior of these systems should be implemented using declarative programming languages that manipulate the logical structures.

We now come to the specific field of service-oriented computing. There are many formal models, which all adhere to at least one of the following dual perspectives: the logic-oriented perspective considering dedicated logics and (labeled) transition systems as models, and the process-oriented perspective considering process calculi or algebra and labeled transition systems as semantic interpretations. These models have been proposed with the aim of capturing aspects of service-oriented computing, following different points of view:

- verification or modeling, with models using automata, e.g. [14], process algebra, e.g. [10,31], or Petri nets, e.g. [29], and associated to verification tools,
- formalization and programming, with models using process calculi, either original like Orc [23] or extending some standard process calculi like the π-calculus [4,27]. [1]

The latter point of view is directly related to our approach. The main difference between these works and ours stands in the starting point. First of all, the process calculi are proposed to provide a good syntax for an orchestration language, allowing features specific to services to be expressed: sessions [4,6] or correlations [37,26,21], compensations (after the abort of a transaction) [27]. On the contrary, we propose a semantic framework, which is minimal. Thanks to impure Criojo, and the Black Box principle, we should be able to integrate these process calculi into our framework. Thanks to pure Criojo, we should be able to encode the specific features of these calculi, by following a discipline of programming, as shown for sessions for instance.

Finally, these theoretical foundations are still a preliminary work. From the theoretical side, first, the development of the theory of bisimulation can lead to interesting extensions with firewalls. Second, if we have shown the computability limitations of the standard chemical abstract machines without introspection, we have not yet formalized a precise notion of computability for chemical abstract machines. This formalization asks for a computability theory over multisets, as developed by Tucker and Zucker for abstract data types [35]. From the practical side, we need for Criojo a proof of concept in order to get a real programming language. An implementation is currently developed, using the language Scala and the technology of Web Services. The complexity of the algorithms involved is a major question, in order to get a clear and precise model of the run time of Criojo programs.

[1] For more references, see a survey from 2007 [34].

References

1. Peter Alvaro, Tyson Condie, Neil Conway, Khaled Elmeleegy, Joseph M. Hellerstein, and Russell Sears. Boom analytics: exploring data-centric, declarative programming for the cloud. In *EuroSys 2010*, pages 223–236, 2010.
2. Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.
3. Hariolf Betz, Frank Raiser, and Thom Frühwirth. A complete and terminating execution model for Constraint Handling Rules. In *ICLP 2010*.
4. Michele Boreale, Roberto Bruni, and Luís Caires et al. SCC: A service centered calculus. In *WS-FM 2006*, volume 4184 of *LNCS*, pages 38–57. Springer-Verlag.
5. Pietro Braione and Gian Pietro Picco. On calculi for context-aware coordination. In *COORDINATION 2004*.
6. Luís Caires and Hugo Torres Vieira. Conversation types. *Theoretical Computer Science*, 411(51–52):4399–4440, 2010.
7. Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1:146–166, 1989.
8. Evgeny Dantsin and Andrei Voronkov. Expressive power and data complexity of query languages for trees and lists. In *PODS 2000*, pages 157–165.
9. Olivier Danvy and Lasse Nielsen. Refocusing in reduction semantics. Technical report, BRICS Report Series, 2004.
10. Andrea Ferrara. Web services: a process algebra approach. In *ICSOC 2004*, pages 242–251.
11. Cédric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *POPL 1996*, pages 372–385.
12. Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *CONCUR 1996*, pages 406–421.
13. Thom Frühwirth. Welcome to constraint handling rules. In *Constraint Handling Rules*, volume 5388 of *LNCS*, pages 1–15. Springer-Verlag, 2008.
14. Xiang Fu, Tevfik Bultan, and Jianwen Su. Analysis of interacting BPEL web services. In *WWW 2004*, pages 621–630, 2004.
15. Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.
16. Harald Ganzinger and David McAllester. Logical algorithms. In *ICLP 2002*, pages 209–223.
17. Deepak Garg, Akash Lal, and Sanjiva Prasad. Effective chemistry for synchrony and asynchrony. In *TCS 2004*.
18. Allen Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47(1):185–221, 1993.
19. Allen Van Gelder, Kenneth Ross, and John Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
20. Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. 1989.
21. Claudio Guidi, Roberto Lucchi, Roberto Gorrieri, Nadia Busi, and Gianluigi Zavattaro. SOCK: a calculus for service oriented computing. In *ICSOC 2006*, pages 327–338.
22. M.N. Huhns and M.P. Singh. Service-oriented computing: key concepts and principles. *Internet Computing, IEEE*, (1):75 – 81, 2005.
23. David Kitchin, William Cook, and Jayadev Misra. A language for task orchestration and its semantic properties. In *CONCUR 2006*, pages 477–491.

24. Mayleen Lacouture, Hervé Grall, and Thomas Ledoux. CREOLE: a Universal Language for Creating, Requesting, Updating and Deleting Resources. In *FOCLASA 2010*.
25. Leslie Lamport and Nancy A. Lynch. Distributed computing: Models and methods. In *Handbook of Theoretical Computer Science, Volume B*, pages 1157–1199. 1990.
26. Alessandro Lapadula, Rosario Pugliese, and Francesco Tiezzi. A calculus for orchestration of web services. In *ESOP 2007*, pages 33–47.
27. Roberto Lucchi and Manuel Mazzara. A pi-calculus based semantics for WS-BPEL. *Journal of Logic and Algebraic Programming*, 70(1):96–118, 2007.
28. Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.
29. Chun Ouyang, Eric Verbeek, Wil van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur ter Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Science of Computer Programming*, 67(2–3):162–198, 2007.
30. Gordon Plotkin. Dijkstra's predicate transformers and Smyth's powerdomains. In *Abstract Software Specifications*, pages 527–553, 1980.
31. Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. Describing and reasoning on web services using process algebra. In *ICWS 2004*, pages 43–50.
32. Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2003.
33. Robert Simmons and Frank Pfenning. Linear logical algorithms. In *ICALP 2008*, pages 336–347.
34. Maurice ter Beek, Antonio Bucchiarone, and Stefania Gnesi. Formal methods for service composition. *Annals of Mathematics, Computing and Teleinformatics*, 1(5):1–10, 2007.
35. John Tucker and Jeffery Zucker. *Computable functions and semicomputable sets on many-sorted algebras*, pages 397–525. Oxford University Press, 2000.
36. Christian Urban, Andrew Pitts, and Murdoch Gabbay. Nominal unification. *Theoretical Computer Science*, 2004.
37. Mirko Viroli. A core calculus for correlation in orchestration languages. *Journal of Logic and Algebraic Programming*, 70(1):74–95, 2007.
38. Peter Van Weert, Jon Sneyers, Tom Schrijvers, and Bart Demoen. Extending CHR with negation as absence. In *CHR 2006*.